Федеральное государственное бюджетное учреждение науки
Центр информационных технологий в проектировании
Российской академии наук

На правах рукописи

Лебедев Артем Сергеевич

Методы и средства распараллеливания программ линейного класса для выполнения на многопроцессорных вычислительных системах

Специальность 2.3.5. Математическое и программное обеспечение вычислительных систем, комплексов и компьютерных сетей

Диссертация на соискание учёной степени
кандидата технических наук

Научный руководитель:
кандидат технических наук
Солодовников Владимир Игоревич

Одинцово — 2024

Оглавление

Стр.

Стр.

Стр.

Стр.

Введение

Повсеместное распространение многоядерных процессоров в мобильных устройствах, рабочих станциях, серверных решениях сделало концепцию параллелизма массовой и доступной. Практика последних лет показала эффективность специализированных вычислителей и ускорителей на основе программируемых логических интегральных схем (ПЛИС) и графических процессорных устройств (ГПУ) применительно к высокопроизводительным вычислениям, однако вычислительные системы, построенные на основе универсальных многоядерных процессоров, в частности вычислительные машины с архитектурой NUMA (Non Uniform Memory Access) и построенные на их основе кластеры, остаются наиболее распространенными и востребованными благодаря их универсальности, наличию развитых инструментов программирования и многообразию библиотек функций.

Эффективное программирование параллельных архитектур всегда было сложной задачей, и особенно усложняется при возрастающих требованиях к быстродействию современного программного обеспечения с объемной кодовой базой, созданной в процессе многолетней разработки. Кроме того, специалисты предметных областей, разрабатывающие программный код для проведения собственных исследований, предпочитают фокусироваться на решении прикладной задачи, а не на технических аспектах параллельного программирования конкретной вычислительной системы, что может повлечь неэффективное использование ресурсов оборудования. Одним из подходов к решению обозначенных проблем является автоматическое распараллеливание программ, исключающее усилия программиста по анализу потока данных в программе, выявлению параллелизма, синтезу параллельного вычислительного кода. Задача автоматического распараллеливания программного кода была сформулирована с момента появления первых параллельных отечественных вычислителей (например, ПС2000). К настоящему времени разработаны языки, модели и инструменты программирования, которые упрощают труд разработчика (DVM-система, САПФОР, НОРМА, OPS, Т-система, Erlang, Go), но не делают распараллеливание автоматическим.

Наибольшая вычислительная трудоемкость сосредоточена в циклических конструкциях. Особый интерес для исследователей представляют программы линейного класса [1, с. 340], встречающиеся в научных и инженерных приложениях,

которые тратят значительную часть времени именно на исполнение гнезд циклов. Модель многогранников [2, с. 1581–1592] предоставляет мощный математический аппарат, упрощающий анализ и преобразование таких программ с целью улучшения их быстродействия путем распараллеливания гнезд циклов и улучшения локальности использования данных [1, с. 57; 3, с. 911, 920, 1039] при вычислениях. Методы модели многогранников развивались в исследовательских проектах LooPo, PIPS, Pluto, PPCG, C-to-CUDA и были применены разработчиками компиляторов в компонентах GCC Graphite, LLVM Polly. Несмотря на обилие работ в этом направлении, задача построения оптимальных преобразований, улучшающих быстродействие программы, до конца не решена. Поэтому тема настоящей диссертационной работы, связанной с разработкой методов автоматического распараллеливания линейных программ для вычислительных систем, построенных на основе универсальных многоядерных процессоров, является востребованной и актуальной.

В диссертационном исследовании был выполнен обзор существующих методов и средств преобразования линейных программ для распараллеливания гнезд циклов. Теоретическую базу диссертации в области методов выпуклого анализа и линейного целочисленного программирования составили работы таких ученых, как Черникова Н.В. (алгоритм для нахождения общей формулы неотрицательных решений системы линейных неравенств), Филипп Клаусс (метод подсчета точек с целочисленными координатами внутри многогранника через полиномы Эрхарта), Манфред Падберг и Джованни Ринальди (метод ветвей и отсечений), в области автоматического распараллеливания программ — Воеводин В.В. и Воеводин Вл.В. (анализ программ линейного класса), Кристиан Ленгауэр (модель многогранников), Поль Футриер (таймирование), Удай Бондхугула (локальность использования данных), Мартин Грибль (распределение операций и данных между процессорами), Седрик Бастуль (кодогенерация). Задачи разбиения операций программы на зерна вычислений и построения распараллеливающих трансляторов рассматривались в работах Левченко В.Д., Перепелкиной А.Ю., Крюкова В.А., Бахтина В.А. Отдельные вопросы прогнозирования времени выполнения вычислений рассматривались в работе Ларкина Е.В. и Ивутина А.Н. [4]. В настоящей диссертации развиты методы П. Футриера, М. Грибля, У. Бондхугулы для нахождения пространственных и временных отображений [5, с. 48] программ линейного класса, ориентированные на рас-

параллеливание гнезд циклов вместе с улучшением локальности использования данных.

Целью работы является повышение быстродействия [6, с. 39] программ, получаемых в результате применения средств автоматического распараллеливания.

Объектом исследований является транслятор, выполняющий автоматическое распараллеливание программ линейного класса для многопроцессорных вычислительных систем.

Предметом исследований являются методы нахождения пространственных и временных отображений программ линейного класса для организации их параллельного выполнения на многопроцессорных вычислительных системах.

Научная задача, решаемая в диссертации — разработка методов нахождения пространственных и временных отображений программ линейного класса, обеспечивающих локальность использования данных при их параллельном выполнении на многопроцессорных вычислительных системах.

Для достижения поставленной цели необходимо решить следующие **задачи**:

1. Провести анализ существующих методов и средств нахождения пространственных и временных отображений программ линейного класса, ориентированных на распараллеливание гнезд циклов.
2. Разработать критерии оптимальности пространственных и временных отображений программ линейного класса с точки зрения локальности использования данных.
3. Разработать метод нахождения оптимальных пространственных и временных отображений программ линейного класса для организации их параллельного выполнения на многопроцессорных вычислительных системах.
4. Разработать метод генерации параллельной MPI-программы, позволяющий организовать информационный обмен между параллельными процессами в случае явно заданного распределения данных между процессорами.
5. Провести анализ быстродействия параллельных программ, полученных применением разработанных методов и средств к тестам из набора PolyBench.

Методология и методы исследования. В работе применяются методы теории множеств, теории графов, линейной алгебры, выпуклого анализа, дискретного программирования, описательной статистики. Практические исследования проведены на кластерной вычислительной системе.

Научная новизна диссертации заключается в том, что в ней разработаны:

1. новые критерии оптимальности пространственных и временных отображений программ линейного класса, **отличающиеся** возможностью ранжировать информационные зависимости и доступы к данным *для более гибкого количественного описания локальности использования данных*;
2. новый метод нахождения оптимальных пространственных и временных отображений программ линейного класса, **отличающийся** применением взвешенной суммы показателей качества решения *для повышения производительности программ при параллельном выполнении*;
3. новый метод генерации параллельной MPI-программы, **не требующий** дублирования входных данных во всех исполняющихся процессах *для сокращения накладных расходов памяти на поддержку распределенных вычислений*.

Положения, выносимые на защиту:

1. Критерии оптимальности пространственных и временных отображений программ линейного класса, предоставляющие более гибкое количественное описание локальности использования данных по сравнению с целевыми функциями на основе лексикографического упорядочивания.
2. Метод нахождения оптимальных пространственных и временных отображений программ линейного класса, позволяющий избежать полного перебора решений с индивидуальной трудоемкой оценкой их качества.
3. Метод генерации параллельной MPI-программы, позволяющий организовать информационный обмен между параллельными процессами в случае явно заданного распределения данных между процессорами, не требуя дублирования входных данных во всех исполняющихся процессах.

Теоретическая значимость работы состоит в развитии подходов к исследованию влияния локальности использования данных на быстродействие программ линейного класса при их параллельном выполнении. Эти подходы могут быть применены при разработке методов и методик повышения быстродействия

программ, затрачивающих большую часть времени выполнения на участки с циклическими конструкциями.

Практическая значимость работы заключается в разработанных компонентах, которые могут быть использованы в автоматически распараллеливающем трансляторе для поддержки распараллеливания программ, написанных на языке Си: компонент нахождения пространственных и временных отображений программ линейного класса, скрипт расстановки директив OpenMP, библиотека макросов для организации информационного обмена и скрипт постобработки параллельных циклов для реализации блочной схемы распределения процессоров в MPI-программе.

Реализация и внедрение результатов работы. Разработанное автором программное обеспечение для распараллеливания программ линейного класса внедрено ООО «НПП САТЭК плюс», а полученные теоретические результаты использованы в учебном процессе Федерального государственного бюджетного образовательного учреждения высшего образования «МИРЭА — Российский технологический университет», что подтверждено соответствующими актами.

Отдельные направления диссертационного исследования были поддержаны грантом РФФИ №14-37-50316 мол_нр «Исследование и разработка методов автоматического распараллеливания программ для гетерогенных вычислительных систем и систем с распределенной памятью» (2014 г.) и грантом Фонда содействия развитию малых форм предприятий в научно-технической сфере (Фонда содействия инновациям) в рамках проекта «Разработка платформы автоматического распараллеливания программ для гетерогенных высокопроизводительных вычислительных систем» (договор №1546ГС1/24323 от 28.09.2016).

Апробация результатов работы. Результаты диссертации докладывались и обсуждались на конференциях: Третий Национальный Суперкомпьютерный Форум (НСКФ-2014) (Переславль-Залесский, Институт программных систем имени А.К. Айламазяна Российской академии наук, 2014), Пятый Национальный Суперкомпьютерный Форум (НСКФ-2016) (Переславль-Залесский, Институт программных систем имени А.К. Айламазяна Российской академии наук, 2016), 11th IEEE International Conference on Application of Information and Communication Technologies (AICT) (Moscow, V.A. Trapeznikov Institute of Control Sciences of Russian Academy of Sciences, 2017), 3rd International Conference «Futuristic Trends in Networks and Computing Technologies» (FTNCT) (Taganrog, Southern Federal University, 2020), Всероссийская научно-техническая конференция «Многопро-

цессорные вычислительный и управляющие системы» (МВУС-2022) (Таганрог, Южный федеральный университет, 2022).

Достоверность и обоснованность научных результатов, полученных соискателем, подтверждены корректностью и непротиворечивостью математических выкладок и доказательств, результатами машинных экспериментов, а также внедрениями и использованием в различных организациях, что подтверждается соответствующими актами.

Публикации. Основные результаты по теме диссертации изложены в 12 печатных работах общим объемом 11,5 п.л., авторский вклад 9,9 п.л., 7 из которых опубликованы в рецензируемых научных журналах, входящих в Перечень ВАК РФ, 2 — в сборниках трудов конференций, индексируемых Web of Science и Scopus, 3 — в иных сборниках тезисов докладов. Зарегистрированы 2 программы для ЭВМ.

Соответствие паспорту специальности. Диссертация соответствует п. 8 («Модели и методы создания программ и программных систем для параллельной и распределенной обработки данных, языки и инструментальные средства параллельного программирования») в части «методов создания программ для параллельной и распределенной обработки данных» и «инструментальных средств параллельного программирования» паспорта специальности 2.3.5. Математическое и программное обеспечение вычислительных систем, комплексов и компьютерных сетей, технические науки.

Объем и структура работы. Диссертация состоит из введения, 4 глав, заключения и 7 приложений. Полный объём диссертации составляет 297 страниц, включая 39 рисунков и 44 таблицы. Список литературы содержит 124 наименования.

Глава 1. Распараллеливание программ в модели многогранников

1.1 Подходы к распараллеливанию гнезд циклов

Основная идея распараллеливания циклов основывается на том, что независимые итерации цикла могут быть выполнены параллельно. Классические методы распараллеливания [7—9] модифицируют текстовое представление структуры циклов в программе до состояния, в котором некоторые циклы становятся параллельными. Этот подход называется распараллеливанием на основе текста и подразумевает применение следующих преобразований: расщепление циклов, слияние циклов, переиндексирование (добавление константы к индексу цикла), масштабирование (умножение индекса цикла на константу), реверс индекса цикла, перестановка порядка циклов, сдвиг циклов [7—12]. Сложность практического применения подхода состоит в правильном выборе набора этих преобразований и порядка их применения. Наиболее успешным отечественным проектом, базирующимся на применении таких преобразований, является Открытая распараллеливающая система [13—15], реализующая интерактивное взаимодействие с пользователем. Система САПФОР [16; 17] выполняет статический анализ циклов для выбора кандидатов на распараллеливание и организует размещение данных на вычислительном оборудовании средствами DVM [18], также предоставляя диалоговую оболочку, упрощающую анализ циклов. Предметно-ориентированные языки, такие как НОРМА [19; 20] (трафаретные вычисления) и Halide [21] (обработка изображений) являются декларативными, и потому их реализации обходятся без статического анализа циклов, используя сведения о параллелизме операций над памятью из предметной области.

Распараллеливание на основе модели — другой, не привязанный к конкретной предметной области, подход, игнорирующий особенности текстового представления программы, и использующий преобразования в рамках определенной математической модели. Если модель обладает достаточной степенью общности, то ожидаемое преимущество по сравнению с распараллеливанием на основе текста заключается в том, что корректная последовательность шагов распараллеливания может быть представлена одним преобразованием, и это преобразование может быть построено с помощью классических математических

методов оптимизации [5]. Некоторые методы в рамках этого подхода чувствительны к структуре вложенности циклов. Например, гнездо циклов называется плотно вложенным тогда и только тогда, когда тело каждого цикла состоит либо из единственного плотно вложенного гнезда циклов, либо (в случае наиболее глубоко вложенного цикла) последовательности инструкций без цикла. Иначе, гнездо циклов называется неплотно вложенным [12].

Оригинальный фреймворк для автоматического распараллеливания гнезд циклов, в дальнейшем названный моделью многогранников (polytope model), был предложен К. Ленгауэром [22]. Были заявлены следующие ограничения:

– допускается только плотно вложенное гнездо циклов, в теле которого допустимы только присваивания;
– все границы циклов должны быть аффинными выражениями, зависящими от индексов обрамляющих циклов и от внешних переменных программы, то есть символических констант, которые обычно представляют размер задачи;
– единственная поддерживаемая структура данных — массив; индексы массива должны быть аффинными функциями, зависящими от индексов обрамляющих циклов и от внешних переменных программы; скаляры рассматриваются как 0-мерные массивы;
– зависимости в исходной программе должны быть однородными, то есть одинаковыми на всех итерациях тела гнезда цикла.

Современное состояние модели многогранников допускает неплотно вложенные гнезда циклов, а также аффинные зависимости. В настоящей работе модель многогранников рассматривается как модель последовательных и параллельных вычислений, применимая для распараллеливания программ, принадлежащих линейному классу [1, с. 340]. Эти программы удовлетворяют следующим ограничениям:

– в программе может использоваться любое число простых переменных и переменных с индексами;
– единственным типом исполнительного оператора может быть оператор присваивания, правая часть которого является арифметическим выражением;
– все повторяющиеся операции описываются только с помощью циклов DO языка Фортран (либо эквивалентными циклами for языка Си); структура вложенности циклов может быть произвольной; шаги изменения пара-

метров циклов всегда равны +1; если у цикла нижняя граница больше верхней, то цикл не выполняется;

– допускается использование любого числа условных и безусловных операторов перехода, передающих управление вниз по тексту; не допускаются побочные выходы из циклов;
– все индексные выражения переменных, границы изменения параметров циклов и условия передачи управления задаются неоднородными формами, линейными по совокупности параметров циклов и внешних переменных программы; все коэффициенты линейных форм являются целыми числами;
– внешние переменные программы всегда целочисленные; конкретные значения внешних переменных известны только перед началом работы программы и неизвестны в момент ее исследования.

В западной литературе программы, удовлетворяющие перечисленным ограничениям, называются SCoP (static control part), что отражает присущую им статическую природу потока управления. Вопросы применения методов модели многогранников при ослаблении перечисленных ограничений обсуждаются в работах [23—26]. В настоящей работе ограничения зафиксированы и используется термин «линейная программа».

Первая задача в этом подходе, основанном на модели, — определить для каждого вычисления, когда оно должно выполняться. Полученное временное распределение называется *расписанием вычислений*. Параллелизм выражается в исполнении множества вычислительных операций в рамках одного временного шага. Ограничениями для такого планирования вычислений являются информационные зависимости между операциями. Кроме расписания существует также пространственное распределение — *размещение вычислений*, определяющее для каждого вычисления, где оно должно происходить. Подобное распределение, называемое *размещением данных*, может быть задано и для элементов данных, определяя их расположение.

1.2 Базовые понятия модели многогранников

Обобщенный граф зависимостей — это ориентированный мультиграф, каждая вершина которого представляет множество соответственных операций (динамических экземпляров одной и той же инструкции) [27]. Каждая операция принадлежит только одной вершине. Ребро графа представляет временное ограничение на две операции, соответствующие начальной и конечной вершине (принцип причинности для информационно зависимых операций u и v: $u \to v \Rightarrow \theta(v) > \theta(u)$, где θ — логическое время). В обобщенном графе зависимостей могут быть петли и циклы. Каждая вершина помечается описанием соответствующих операций, а каждое ребро — описанием соответствующих отношений зависимости. Пример программы LU-разложения квадратной матрицы и обобщенный граф зависимостей проиллюстрированы на рисунке 1.1. В аннотациях к ребрам слева от стрелки указан доступ к массиву, который, согласно расписанию, должен произойти раньше, а справа — позже, чтобы не нарушить информационную зависимость, и, соответственно, корректность вычислений. Цветовая индикация соответствует трем видам информационных зависимостей: красный — чтение после записи (истинная зависимость), зеленый — запись после чтения (антизависимость), синий — запись после записи (зависимость через выход).

```
1 for (int k = 0; k < N; k++) {
2   for (int l = k + 1; l < N; l++)
3     A[l][k] /= A[k][k]; //S0
4   for (int i = k + 1; i < N; i++)
5     for (int j = k + 1; j < N; j++)
6       A[i][j] -= A[i][k] * A[k][j]; //S1
7 }
```

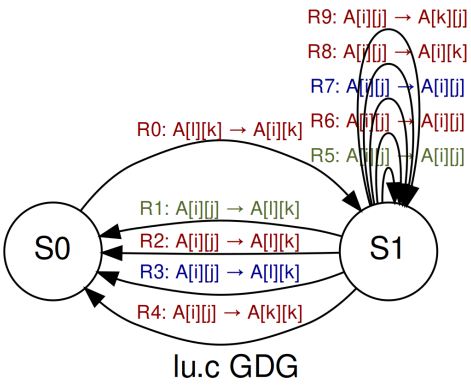
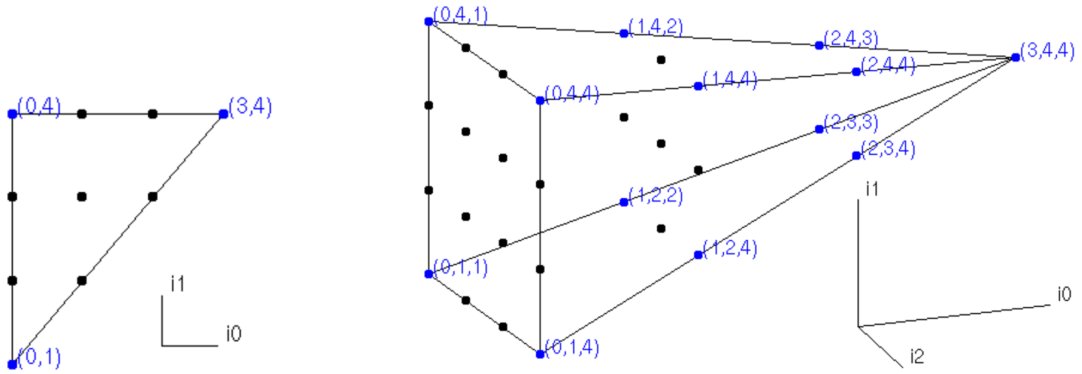


Рисунок 1.1 — LU-разложение квадратной матрицы и обобщенный граф зависимостей программы.

Каждой вершине X соответствует ее домен — многогранник D_X на множестве $\mathbb{Q}^{p_X}$, где p_X — размерность ее пространства итераций (количество циклов в программе, включающих X). На рисунке 1.2 проиллюстрированы многогранники, соответствующие доменам инструкций программы LU-разложения для

размера задачи $N = 5$. Для домена D_{S_0} измерения `i0`, `i1` соответствуют циклам по `k`, `l`; для домена D_{S_1} измерения `i0`, `i1`, `i2` соответствуют циклам по `k`, `i`, `j`.



а) Домен D_{S_0} б) Домен D_{S_1}

Рисунок 1.2 — Домены инструкций программы LU-разложения при $N = 5$.

Каждая операция представляется как $\langle \vec{i}, \vec{z}; X \rangle$, где $\vec{i} \in D_X$ — целочисленный вектор индекса итерации, $\vec{z}$ — целочисленный вектор внешних переменных программы, имеющий размерность q_z. Априорная информация о внешних переменных программы, включающая возможные ограничения, называется *контекстом*. Все операции над памятью, которые требуется выполнить, формируют множество Ω. Каждому ребру e, исходящему из вершины $\sigma(e) = X$ в $\delta(e) = Y$, ставится в соответствие многогранник R_e на множестве $\mathbb{Q}^{p_X + p_Y}$ такой, что условие причинности должно быть наложено на операции $\langle \vec{i}, \vec{z}; X \rangle$ и $\langle \vec{j}, \vec{z}; Y \rangle$ тогда и только тогда, когда составной вектор $\begin{bmatrix} \vec{i} \\ \vec{j} \end{bmatrix} \in R_e$.

Формально обобщенный граф зависимостей представляется четверкой $\langle V; E; \mathcal{D}; \mathcal{R} \rangle$, где V — множество вершин, E — множество ребер, $\mathcal{D}$ — функция из множества вершин V во множество соответствующих доменов, $\mathcal{R}$ — функция из множества ребер E во множество соответствующих многогранников зависимостей [27].

Пусть f и g — индексные функции ячейки памяти, к которой обращаются конфликтующие операции, тогда p_e называют глубиной зависимости [27]:

$$\begin{bmatrix} \vec{x} \\ \vec{y} \end{bmatrix} \in R_e \equiv f(\vec{x}) = g(\vec{y}) \wedge (\vec{x}[1..p_e] = \vec{y}[1..p_e]) \wedge (\vec{x}[p_e + 1] < \vec{y}[p_e + 1]). \quad (1.1)$$

d-мерное ($d \geqslant 1$) расписание вычислений для обобщенного графа зависимостей есть функция $\theta\colon \Omega \to \mathbb{N}_0^d$ такая, что

$$\forall e \in E, \vec{x} \in D_{\sigma(e)}, \vec{y} \in D_{\delta(e)}\colon \begin{bmatrix}\vec{x}\\ \vec{y}\end{bmatrix} \in R_e \Rightarrow \theta\left(\langle \vec{y},\vec{z};\delta(e)\rangle\right) >_{lex} \theta\left(\langle \vec{x},\vec{z};\sigma(e)\rangle\right). \tag{1.2}$$

Множество операций $F(t) = \{u \in \Omega | \theta(u) = t\}$ называется фронтом расписания на t (или ярусом параллельной формы [1, с. 194]). Программа с синхронным параллелизмом может быть сконструирована следующим образом:

```
1 do t = 0; L
2   exchange data
3   pardo F(t)
4   barrier
5 end do
```

Здесь $L = \max\limits_{u\in\Omega}(\theta(u))$ — задержка расписания [27].

Размещение вычислений есть функция $\pi\colon \Omega \to \mathbb{N}_0$, которая ставит в соответствие операции номер виртуального процессора, на котором она будет исполняться. Пространство виртуальных процессоров, как правило, предполагается бесконечным [3, с. 925]. Пусть $\langle g_a, \vec{z}; A_a\rangle$ — доступ к массиву A_a с аффинной индексной функцией $g_a(\vec{i}), \quad \vec{i} \in D_X$ в некоторой позиции a инструкции X. Тогда $\eta(\langle g_a, \vec{z}; A_a\rangle)$ — номер виртуального процессора, на котором располагается элемент $A_a[g_a(\vec{i})], \quad \vec{i} \in D_X$. Сопоставление виртуальных процессоров физическим может выполняться согласно различным схемам [28, с. 106]. Среди самых распространенных схем отмечают:

1. блочная схема: $r(v) = \lfloor \frac{v}{B} \rfloor$, где v — номер виртуального процессора, $r(v)$ — номер сопоставленного v физического процессора, B — размер блока;
2. циклическая схема: $r(v) = v \mod Q$, где Q — количество физических процессоров.

На рисунке 1.3 проиллюстрированы рассматриваемые аффинные отображения для двух массивов и двух инструкций. Символом «*» обозначен элемент массива, вовлеченный в информационную зависимость.

Назначением процедуры анализа потока данных [29] является упрощение представления R_e. Для каждой зависимости e имеем многогранник P_e и аффинное отображение h_e такое, что:

$$\begin{bmatrix}\vec{x}\\ \vec{y}\end{bmatrix} \in R_e \equiv (\vec{x} = h_e(\vec{y}) \wedge \vec{y} \in P_e).$$

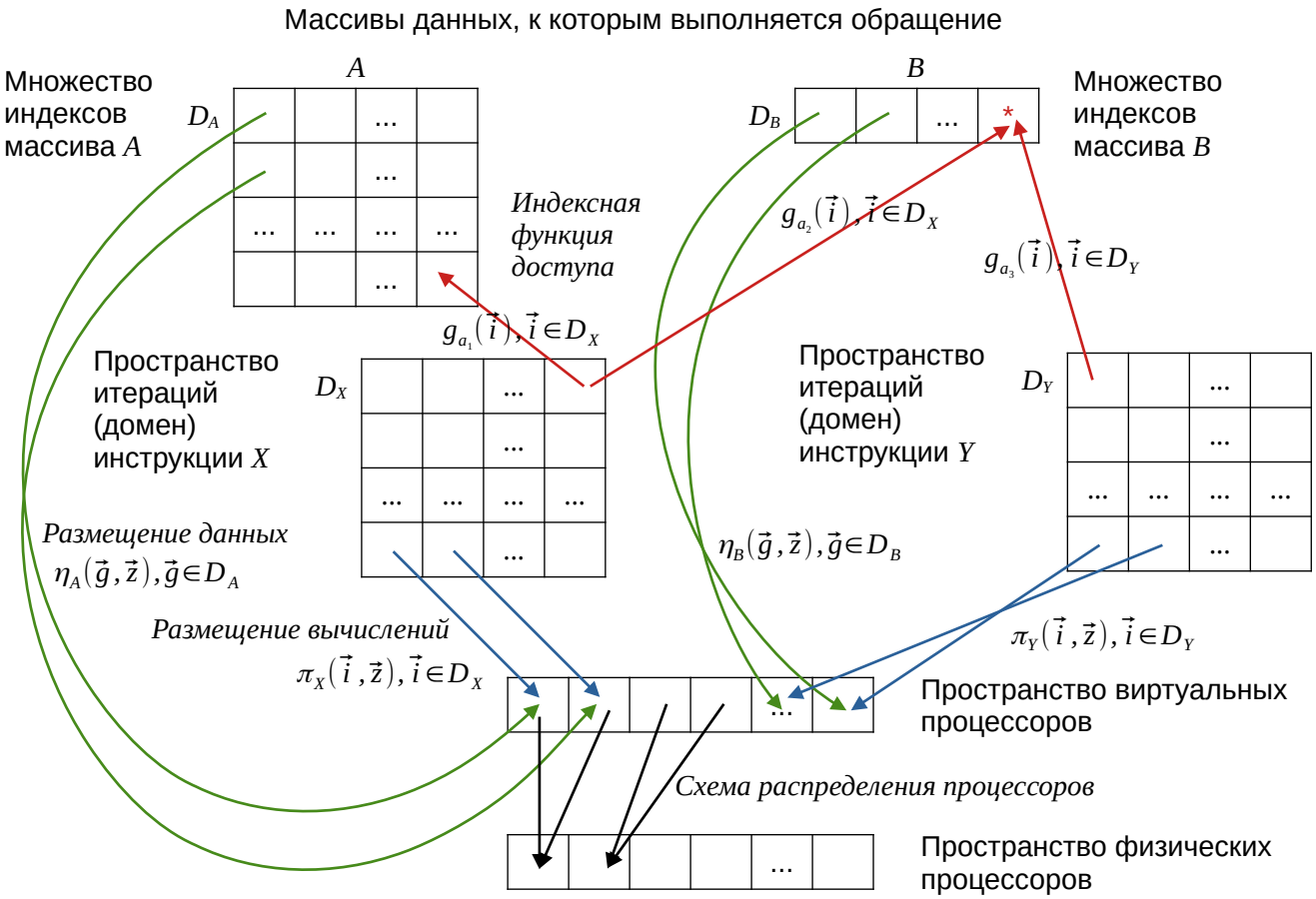


Рисунок 1.3 — Аффинные отображения в модели многогранников.

1.3 Этапы распараллеливания линейной программы

1.3.1 Анализ зависимостей по данным

Выделяют два вида тестов на зависимость по данным: точные и приближенные [30—32]. Платой за точное решение служит более низкое быстродействие теста. В случае статической компиляции кода проблема низкого быстродействия не стоит так остро, как для JIT-трансляции, поэтому точные тесты развивались в направлении применения методов математического программирования, и нашли применение в трансляторах source-to-source (текст-в-текст). Разработанный Футриером [29] метод дает точное решение задачи анализа потока данных благодаря использованию аппарата параметрического целочисленного программирования [33] для решения задачи лексикографической оптимизации. Этот метод был расширен Коллардом и Гриблем [34] для анализа достигающих опре-

делений и реализован в проекте LooPo [35], разработка которого прекращена. Дальнейшее развитие метод Футриера получил в направлении применения к программам с динамическим потоком управления [25; 26]. Оригинальный метод Футриера реализован в проекте candl [36], разработка которого продолжается, а компоненты программного обеспечения используются в сторонних проектах, таких как транслятор pluto [37; 38], и в разработанной автором системе ilpy [39—43].

1.3.2 Нахождение расписания вычислений

Метод гиперплоскостей Лампорта [44] считается одним из первых основанных на модели методов автоматического распараллеливания циклов [5, с. 18]. Он был применим только для программ с однородными зависимостями. Футриером [27] был предложен метод построения одномерных расписаний, оптимизирующих временную локальность использования данных (bounded delay schedule). Для каждой информационной зависимости задавалось ограничение:

$$\begin{bmatrix}\vec{x}\\ \vec{y}\end{bmatrix} \in R_e \Rightarrow 1 \leqslant \theta\left(\langle\vec{y},\vec{z};\delta(e)\rangle\right) - \theta\left(\langle\vec{x},\vec{z};\sigma(e)\rangle\right) \leqslant \Delta, \tag{1.3}$$

где Δ — целочисленная константа. Не всякий обобщенный граф зависимостей позволяет найти расписание, удовлетворяющее этим ограничениям. В общем случае, не для всякой программы существует одномерное расписание вычислений. Футриером был предложен жадный алгоритм, минимизирующий размерность многомерного расписания [45]. В дальнейшем он был реализован в трансляторе PIPS [46]. Бондхугулой были развиты идеи Футриера оптимизации временной локальности использования данных, и реализованы без минимизации размерности расписания в распараллеливающем трансляторе pluto [37] с использованием лексикографической оптимизации при поиске решений [33]. Для каждой инструкции вычисляется столько же легальных отображений φ, сколько их существует в исходной программе:

$$\begin{bmatrix}\vec{x}\\ \vec{y}\end{bmatrix} \in R_e \Rightarrow 0 \leqslant \varphi\left(\langle\vec{y},\vec{z};\delta(e)\rangle\right) - \varphi\left(\langle\vec{x},\vec{z};\sigma(e)\rangle\right) \leqslant L_e(\vec{z}), \tag{1.4}$$

где $L_e(\vec{z})$ — аффинная функция, зависящая только от внешних переменных программы. Предполагается минимизация этой верхней границы. При этом все вычисляемые отображения линейно не зависимы.

Подход, предложенный Бондхугулой, демонстрируют эффективность на практике [37; 38; 47; 48]. Он реализован не только в трансляторе pluto, но и в библиотеке isl [49], на основе которой построены решения LLVM Polly [50; 51] и GCC Graphite [52]. Однако, постановка задачи оптимизации не лишена следующего конструктивного недостатка: поскольку поиск решений сводится к поиску лексикографического минимума на многограннике, возникает вопрос о порядке следования переменных в целевом векторе, то есть об относительной значимости всех зависимостей по данным. Кроме того, в такой постановке задачи оптимизации невозможно задать одинаковый приоритет различным зависимостям, поскольку невозможно поставить две переменные на одну и ту же позицию в целевом векторе. Разработанный в диссертации метод нахождения пространственных и временных отображений [53—58] минимизируют задержку и расстояние использования данных, снимая указанное ограничение.

Одним из направлений дальнейшего развития методов нахождения расписаний является гибридный подход с динамическим планированием [59—62].

1.3.3 Нахождение размещения вычислений и данных

Снижение стоимости коммуникаций в многопроцессорных системах является одним из способов улучшения быстродействия параллельной программы. В работах Лим и Лам [63—65] предлагаются алгоритмы для уменьшения накладных расходов на синхронизацию, в частности рассматриваются случаи параллелизма без синхронизации:

$$\begin{bmatrix}\vec{x}\\ \vec{y}\end{bmatrix} \in R_e \Rightarrow \pi\left(\langle\vec{y},\vec{z};\delta(e)\rangle\right) - \pi\left(\langle\vec{x},\vec{z};\sigma(e)\rangle\right) = 0. \tag{1.5}$$

Для систем с общей памятью решается только задача нахождения размещения вычислений. Бондхугула в проекте pluto продолжает построение линейно независимых отображений, оптимизирующих локальность использования данных, сохраняя размерность пространств итераций инструкций по отношению к

исходной программе. Грибль [5] предлагает полный перебор решений и выбор лучшего с точки зрения исследователя, при этом не приводя формального критерия качества. В отличие от известной схемы Грибля разработанный в диссертации метод [54; 57] устраняет необходимость полного перебора решений с их трудоемкой оценкой качества, и при этом позволяет эффективнее оптимизировать пространственную локальность использования данных по сравнению с методом Бондхугулы благодаря использованию аппарата линейного целочисленного программирования вместо лексикографической оптимизации.

Для систем с распределенной памятью выполняется нахождение размещения вычислений и данных совместно. Футриером предложен жадный алгоритм, нацеленный на полное исключение коммуникаций для как можно большего количества информационных зависимостей [66; 67]. Этот алгоритм был также реализован в трансляторе PIPS [46]. Грибль [5] предлагает вариант полного перебора решений, как и для систем с общей памятью. В отличие от известной схемы Грибля разработанный в диссертации метод [68; 69] устраняет необходимость полного перебора решений с их трудоемкой оценкой качества, и при этом позволяет эффективно оптимизировать пространственную локальность использования данных согласно идее Футриера, но при ослабленных ограничениях, заключающихся в сокращении расстояния коммуникаций, а не полном их исключении. Последнее не является необходимым на практике, так как коммуникация между ядрами многоядерного процессора не является дорогой операцией, что и обуславливает ослабление ограничений. Выбор решений также сводится к задаче линейного целочисленного программирования.

1.3.4 Разбиение операций программы на зерна вычислений (тайлинг)

Итерации циклов логического времени и виртуальных процессоров могут быть агрегированы в блоки для дополнительного улучшения временной и пространственной локальности использования данных. В работе Лим и Лам [70] предложен метод на основе классических преобразований циклов. Систематическое построение гиперплоскостей рассматривается в работе [71]. Современные методы, оптимизированные для трафаретных вычислений, активно разрабатываются в ИПМ имени М. В. Келдыша Российской академии наук [72; 73].

Экспериментальную реализацию в проекте pluto имеют методы [74; 75]. Разбиение для суперкомпьютера с многоуровневой иерархией памяти было предложено в работах [76; 77]. Гибридный вариант разбиения для системы с ускорителем GPU рассмотрен автором в работе [78]. Методы, оптимизированные для целевой архитектуры NVIDIA CUDA, представлены в работах [79—85]. Частные вопросы загрузки тензорных ядер GPU рассматриваются в работе [86], а утилизации SIMD-инструкций центрального процессора — в работе [87].

1.3.5 Генерация кода

Для систем с общей памятью задача генерации параллельной программы исчерпывающе решается методом Бастуля [88], имеющим программную реализацию в проекте cloog, поддерживаемом разработчиками, и используемом в других проектах (pluto, ilpy). Генерируется параллельная программа на языке Си с директивами OpenMP.

Для систем с распределенной памятью задача генерации параллельной программы усложняется необходимостью организации информационного обмена между параллельными процессами. Дататри и соавторами был предложен и реализован в проекте pluto (ветка distmem [89]) метод FOP (flow-out partitioning) [90], определяющий поток данных между итерациями распределенных циклов вдоль векторов информационных зависимостей, и исключающий дублирование информации при пересылке информационных пакетов. При этом входные данные размещаются на всех вычислительных устройствах, и вычисление размещения данных не производится. Разработанный в диссертации метод [91] учитывает оптимальное размещение вычислений и данных, вводит понятие многогранников коммуникаций, и преобразует параллельную программу, сгенерированную по методу Бастуля, в параллельную MPI-программу с двусторонней коммуникацией процессов. При этом не требуется размещать данные на всех вычислительных машинах и применять специальные модели организации вычислений, как это предлагается, например, в работах [92—94].

1.4 Общие выводы по главе

Методы модели многогранников применимы не только для универсальных многоядерных процессоров и построенных на их основе кластерных систем, но и для гетерогенных вычислительных систем, включающих графические ускорители, ПЛИС [95]. Также существуют реализации предметно-ориентированных языков, использующие методы модели многогранников для организации параллельных вычислений: трафаретные вычисления OP2 [96] и машинное обучение OptiML [97; 98] с применением PENCIL [99; 100], вычисления на графах DFGL [101]. В работе [102] рассматривается применение методов машинного обучения для быстрого выбора аффинных отображений на целевом компьютере.

Значительная часть методов модели многогранников в части нахождения аффинных отображений и тайлинга нацелены на улучшение локальности использования данных либо путем сближения операций и операндов во времени, либо в пространстве виртуальных процессоров, либо на физических процессорах при определении зерен вычислений. Разработанный в диссертации метод нахождения пространственных и временных отображений программ линейного класса для распараллеливания гнезд циклов следует этим соображениям. Для его реализации на системах с распределенной памятью также реализован метод организации информационного обмена в рамках стандарта MPI.

Классификация современных методов модели многогранников вместе с разработанными в работе методами проиллюстрирована на рисунке 1.4.

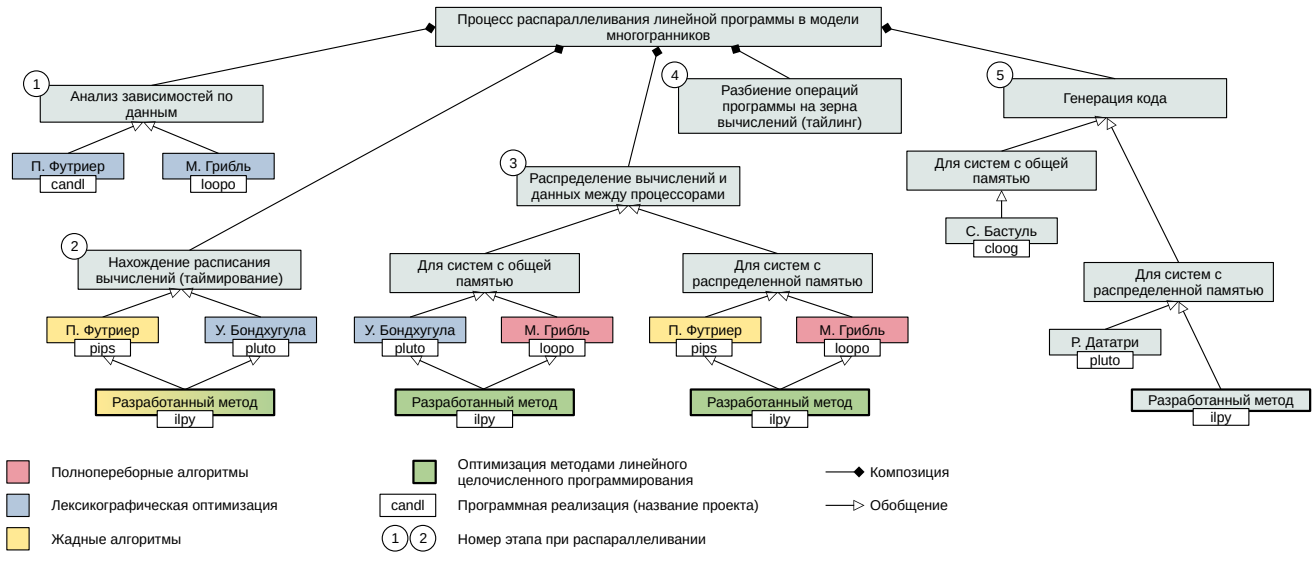


Рисунок 1.4 — Разработанные методы на основе модели многогранников.

Глава 2. Метод нахождения пространственных и временных отображений программ линейного класса для распараллеливания гнезд циклов

2.1 Аффинные отображения. Критерий оптимальности

Одномерное аффинное отображение для инструкции X есть функция вида

$$\varphi_X(\vec{i},\vec{z}) = \vec{v}_X \cdot \vec{i} + \vec{v}'_X \cdot \vec{z} + v^0_X, \quad \vec{i} \in D_X, v^0_X \in \mathbb{Z}, \vec{v}_X \in \mathbb{Z}^{p_X}, \vec{v}'_X \in \mathbb{Z}^{q_z}.$$

Многомерное аффинное отображение для инструкции X задается конечным набором (вектором) одномерных отображений:

$$\Phi_X(\vec{i},\vec{z}) = \left[\varphi^1_X(\vec{i},\vec{z}) \ \ \ldots \ \ \varphi^{q_X}_X(\vec{i},\vec{z})\right]^T, \quad q_X \leqslant p_X.$$

Рассмотрим функцию $\chi_e(\vec{y},\vec{z}) = \varphi_{\delta(e)}(\vec{y},\vec{z}) - \varphi_{\sigma(e)}(h_e(\vec{y}),\vec{z})$, $\vec{y} \in P_e$ как меру повторного использования данных.

Если отображения $\varphi_{\delta(e)}(\vec{y},\vec{z})$ и $\varphi_{\sigma(e)}(h_e(\vec{y}),\vec{z})$ используются для задания логического времени (представляют расписание вычислений), то $\chi_e(\vec{y},\vec{z})$ — количество отсчетов логического времени, проходящего между исполнением двух операций $\langle h_e(\vec{y}),\vec{z},\sigma(e)\rangle$ и $\langle \vec{y},\vec{z},\delta(e)\rangle$, то есть мера задержки использования данных $d^\tau_e(\vec{y},\vec{z}) = \chi_e(\vec{y},\vec{z})$.

Если отображения представляют размещение вычислений, то $\chi_e(\vec{y},\vec{z})$ — расстояние в пространстве виртуальных процессоров межу процессорами, выполняющими две операции $\langle h_e(\vec{y}),\vec{z},\sigma(e)\rangle$ и $\langle \vec{y},\vec{z},\delta(e)\rangle$, то есть мера расстояния использования данных $d^\rho_e(\vec{y},\vec{z}) = \chi_e(\vec{y},\vec{z})$. Если размещение данных задается явно, то мера расстояния использования данных определяется для доступа a в некоторой позиции инструкции X_a к массиву A_a как $d^\rho_a(\vec{i},\vec{z}) = |\pi_{X_a}(\vec{i},\vec{z}) - \eta_{A_a}(g_a(\vec{i}),\vec{z})|, \vec{i} \in D_{X_a}$. $d^\rho_a(\vec{i},\vec{z})$ — расстояние в пространстве виртуальных процессоров межу процессором, выполняющим инструкцию X_a, и процессором, владеющим элементом данных $A_a[g_a(\vec{i},\vec{z})]$, к которому осуществляется доступ a в некоторой позиции инструкции X_a.

Эвристическое соображение состоит в том, что при уменьшении d^τ_e уменьшается вероятность вытеснения данных из кэша к моменту их использования, а при уменьшении d^ρ_e (или d^ρ_a) возрастает вероятность сближения операций и

операндов: в рамках одной вычислительной машины источником повторного использования данных служит оперативная память, доступная всем процессорам; в рамках одного многоядерного процессора источником повторного использования данных служит кэш определенного уровня, доступный всем ядрам, и, в идеальном случае, в рамках одного ядра повторное использование данных может произойти с использованием кеша первого уровня или даже регистров. Сближение операций и операндов позволяет уменьшить стоимость коммуникаций — например, вместо межмашинной пересылки сообщений с использованием сетевых интерфейсов предпочтительнее обойтись межпроцессорной в рамках одной машины.

Оптимальное расписание программы минимизирует целевую функцию

$$C_E^{\tau}(\theta) = \sum_{e \in E} \alpha_e \max_{\vec{y} \in D_{\delta(e)}} d_e^{\tau}(\vec{y}, \vec{z}), \quad \alpha_e > 0, \tag{2.1}$$

где α_e — весовые коэффициенты, рассматриваемые как показатели относительной значимости зависимостей по данным.

Аналогично, оптимальное размещение программы минимизирует целевую функцию

$$C_E^{\rho}(\pi) = \sum_{e \in E} \alpha_e \max_{\vec{y} \in D_{\delta(e)}} d_e^{\rho}(\vec{y}, \vec{z}), \quad \alpha_e > 0. \tag{2.2}$$

В случае явно заданного расположения данных оптимальное размещение вычислений и данных, вычисленное совместно, минимизирует целевую функцию

$$C_{\{a\}}^{\rho}(\pi, \eta) = \sum_{a \in \{a\}} \alpha_a \max_{\vec{i} \in D_{X_a}} d_a^{\rho}(\vec{i}, \vec{z}), \quad \alpha_a > 0, \tag{2.3}$$

где α_a — весовые коэффициенты, рассматриваемые как показатели относительной значимости доступов к данным.

Минимизация задержки и расстояния использования данных приводит к улучшению временной и пространственной локальности использования данных соответственно. Величины d_e^{τ}, d_e^{ρ}, d_a^{ρ} желательно ограничить постоянными значениями. Однако, это не всегда возможно. Разработанный метод нацелен на то, чтобы минимизировать эти величины совместно, при этом имея возможность задать приоритет для каждой: на уровне зависимости по данным e для d_e^{τ} и d_e^{ρ}; на уровне доступа к данным a для d_a^{ρ}.

Ответ на вопрос о верхней границе для d_e^{τ} и d_e^{ρ} дает следующее утверждение.

Утверждение 1. *Если домены $D_{\delta(e)}$ и $D_{\sigma(e)}$ являются ограниченными, и заданы два аффинных отображения $\varphi_{\delta(e)}(\vec{y}, \vec{z})$ и $\varphi_{\sigma(e)}(h_e(\vec{y}), \vec{z})$, то найдется аффинная*

функция $L(\vec{z})$, зависящая только от внешних переменных программы, такая, что

$$L(\vec{z}) - \chi_e(\vec{y}, \vec{z}) \geqslant 0, \quad y \in P_e.$$

Доказательство. Поскольку h-преобразование является аффинным [29], то его можно задать матрицей $\boldsymbol{H}$:

$$h_e(\vec{y}) = \boldsymbol{H}_e \begin{bmatrix} \vec{y} \\ \vec{z} \\ 1 \end{bmatrix}, \quad \boldsymbol{H}_e \in \mathbb{Q}^{p_{\sigma(e)} \times (p_{\delta(e)} + q_z + 1)}$$

Разобьем матрицу $\boldsymbol{H}_e$ на блоки, отдельно участвующие в умножении на $\vec{y}$, $\vec{z}$, 1:

$$h_e(\vec{y}) = \begin{bmatrix} \boldsymbol{H}_e^{\vec{y}} & \boldsymbol{H}_e^{\vec{z}} & \boldsymbol{H}_e^1 \end{bmatrix} \begin{bmatrix} \vec{y} \\ \vec{z} \\ 1 \end{bmatrix} = \boldsymbol{H}_e^{\vec{y}} \vec{y} + \boldsymbol{H}_e^{\vec{z}} \vec{z} + \boldsymbol{H}_e^1$$

Очевидно, что $\chi_e(\vec{y}, \vec{z}) = \vec{v}_{\delta(e)} \cdot \vec{y} - \vec{v}_{\sigma(e)} \cdot (\boldsymbol{H}_e^{\vec{y}} \vec{y} + \boldsymbol{H}_e^{\vec{z}} \vec{z} + \boldsymbol{H}_e^1) + (\vec{v}'_{\delta(e)} - \vec{v}'_{\sigma(e)}) \cdot \vec{z} + v^0_{\delta(e)} - v^0_{\sigma(e)} = (\vec{v}^T_{\delta(e)} - \vec{v}^T_{\sigma(e)} \boldsymbol{H}_e^{\vec{y}}) \vec{y} + (\vec{v}'^T_{\delta(e)} - \vec{v}'^T_{\sigma(e)} - \vec{v}^T_{\sigma(e)} \boldsymbol{H}_e^{\vec{z}}) \vec{z} + v^0_{\delta(e)} - v^0_{\sigma(e)} - \vec{v}^T_{\sigma(e)} \boldsymbol{H}_e^1$.

$P_e \subseteq D_{\delta(e)}$ — ограниченный выпуклый многогранник, заданный параметрически:

$$\vec{y} \in P_e \Leftrightarrow \vec{y} \in D_{\delta(e)} \wedge h_e(\vec{y}) \in D_{\sigma(e)} \Leftrightarrow \begin{cases} \boldsymbol{A}_{\delta(e)} \begin{bmatrix} \vec{y} \\ \vec{z} \end{bmatrix} + \vec{b}_{\delta(e)} \geqslant 0 \\ \boldsymbol{A}_{\sigma(e)} \begin{bmatrix} h_e(\vec{y}) \\ \vec{z} \end{bmatrix} + \vec{b}_{\sigma(e)} \geqslant 0 \end{cases} \Leftrightarrow$$

$$\begin{cases} \boldsymbol{A}_{\delta(e)} \begin{bmatrix} \vec{y} \\ \vec{z} \end{bmatrix} + \vec{b}_{\delta(e)} \geqslant 0 \\ \begin{bmatrix} \boldsymbol{A}^{h_e(\vec{y})}_{\sigma(e)} & \boldsymbol{A}^{\vec{z}}_{\sigma(e)} \end{bmatrix} \begin{bmatrix} \boldsymbol{H}_e^{\vec{y}} \vec{y} + \boldsymbol{H}_e^{\vec{z}} \vec{z} + \boldsymbol{H}_e^1 \\ \vec{z} \end{bmatrix} + \vec{b}_{\sigma(e)} \geqslant 0 \end{cases} \Leftrightarrow$$

$$\begin{cases} \boldsymbol{A}_{\delta(e)} \begin{bmatrix} \vec{y} \\ \vec{z} \end{bmatrix} + \vec{b}_{\delta(e)} \geqslant 0 \\ \boldsymbol{A}^{h_e(\vec{y})}_{\sigma(e)} \boldsymbol{H}_e^{\vec{y}} \vec{y} + (\boldsymbol{A}^{h_e(\vec{y})}_{\sigma(e)} \boldsymbol{H}_e^{\vec{z}} + \boldsymbol{A}^{\vec{z}}_{\sigma(e)}) \vec{z} + \boldsymbol{A}^{h_e(\vec{y})}_{\sigma(e)} \boldsymbol{H}_e^1 + \vec{b}_{\sigma(e)} \geqslant 0 \end{cases}$$

Значит, существует решение задачи линейного программирования $\chi_e(\vec{y}, \vec{z}) \to \max\limits_{\vec{y} \in P_e}$. Максимум $\chi_e(\vec{y}, \vec{z})$ достигается в одной из вершин P_e.

Внешние переменные программы $\vec{z}$ линейно входят в уравнения гиперплоскостей, определяющих грани P_e. Поскольку вершины P_e являются точками пересечения нескольких граней, то их координаты задаются аффинными функциями, зависящими от $\vec{z}$. Следовательно, значение χ_e в любой вершине представимо аффинной функцией, зависящей от $\vec{z}$. Для каждой вершины S с координатами $\vec{s}$ имеем:

$$\exists \boldsymbol{V}_S \in \mathbb{Q}^{p_{\delta(e)} \times q_z}, \vec{v}_S \in \mathbb{Q}^{p_{\delta(e)}}, \vec{u}_S \in \mathbb{Q}^{q_z}, u_S^0 \in \mathbb{Q}(\vec{s} = \boldsymbol{V}_S \vec{z} + \vec{v}_S \Rightarrow \chi_e(\vec{s}, \vec{z}) = \vec{u}_S \cdot \vec{z} + u_S^0).$$

Перебрав все вершины S ограниченного многогранника P_e, можно выбрать любое $L(\vec{z}) = \vec{l} \cdot \vec{z} + l^0$, удовлетворяющее условию:

$$l^0 \geqslant \max_S(u_S^0), \quad \vec{l}[i] \geqslant \max_S(\vec{u}_S[i]), i = 1 \ldots q_z \tag{2.4}$$

□

Из утверждения 1 следует, что можно выбрать аффинную функцию $L_e(\vec{z}) = \vec{l_e} \cdot \vec{z} + l_e^0$ с наименьшими коэффициентами, ограничивающую сверху $\chi_e(\vec{y}, \vec{z})$. Это можно сделать, рассмотрев ограничения в условии (2.4) как равенства.

Определение Пусть $\{\varphi\}_{E'}$ — множество допустимых наборов φ одномерных аффинных отображений φ_X, $X \in \bigcup\limits_{e \in E'} \{\sigma(e), \delta(e)\}$ для заданного подмножества ребер $E' \subseteq E$ обобщенного графа зависимостей. Набор $\varphi \in \{\varphi\}_{E'}$ называется оптимальным для E', если он минимизирует функционалы $f_e(\varphi) = L_e(\vec{z}) = \vec{l_e} \cdot \vec{z} + l_e^0$, $e \in E'$ так, что

$$\nexists \varphi' \in \{\varphi\}_{E'} \left(\forall e \in E' \left(f_e(\varphi') \leqslant f_e(\varphi) \right) \wedge \exists e \in E' \left(f_e(\varphi') < f_e(\varphi) \right) \right).$$

Нахождение оптимального набора подразумевает решение задачи оптимизации, где:

1. $\{\varphi\}_{E'}$ — множество допустимых решений.
2. $f_e(\varphi) = L_e(\vec{z})$, $e \in E'$ — показатели качества решения φ (меньшие значения свидетельствуют о более предпочтительном решении).

Будем искать решение, применив взвешенную сумму:

$$\sum_{e \in E'} \alpha_e f_e \to \min, \quad \alpha_e > 0,$$

где α_e — весовые коэффициенты, рассматриваемые как показатели относительной значимости зависимостей по данным.

Оптимальное решение минимизирует

$$C^{\chi}_{E'}(\varphi) = \sum_{e \in E'} \alpha_e(\vec{l}_e \cdot \vec{z} + l^0_e). \tag{2.5}$$

С одной стороны, оптимальное решение не может быть улучшено ни по какому из показателей качества $L_e(\vec{z}) = \vec{l}_e \cdot \vec{z} + l^0_e$. С другой стороны, каждое полученное $L_e(\vec{z})$ ограничивает сверху $\chi_e(\vec{y}, \vec{z}), \quad y \in P_e$. Значит, $L_e(\vec{z}) = \max\limits_{\vec{y} \in D_{\delta(e)}} \chi_e(\vec{y}, \vec{z})$. Следовательно, оптимальный набор аффинных отображений доставит минимум (2.1) или (2.2) в зависимости от выбранной семантики одномерных аффинных отображений.

Такая формулировка в силу большей гибкости относительно [27; 37; 45] позволяет описать естественную эвристику: улучшение локальности использования данных более значимо для тех зависимостей, которые чаще возникают. Для программ со слабой параметризацией, или в случае JIT-трансляции, когда значения внешних переменных программы становятся известными, становится возможной приоритизация зависимостей на основе частоты возникновения ситуаций доступа к одной ячейке памяти: $\alpha_e = \#P_e$, где $\#P_e$ — количество точек с целочисленными координатами внутри многогранника зависимостей P_e, которое может быть вычислено с помощью алгоритма Клаусса [103]. В случае статической компиляции рекомендуется выбрать значения $\vec{z}$ как можно более близкие к вероятным исходя из априорной информации относительно контекста, а затем вычислить $\#P_e$ в соответствии с предполагаемыми значениями.

Далее формулировка задач для отыскания расписаний и размещений вычислений будет уточняться путем задания набора ограничений. При этом зафиксируем, что обобщенный граф зависимостей анализируемой программы содержит m вершин S_i, $i = 1, \ldots, m$ и n ребер e_j, $j = 1, \ldots, n$.

2.2 Одномерные аффинные расписания вычислений

Пусть $\theta_{S_i}(\vec{x}, \vec{z}), \quad \vec{x} \in D_{S_i}$ — аффинная функция расписания для инструкции S_i. Рассмотрим функцию $\tau_{e_j}(\vec{y}, \vec{z}) = \theta_{\delta(e_j)}(\vec{y}, \vec{z}) - \theta_{\sigma(e_j)}(h_{e_j}(\vec{y}), \vec{z}), \quad \vec{y} \in P_{e_j}$. Эта функция дает меру задержки использования данных, поскольку фронты расписания исполняются поочередно. Минимизация этой задержки приводит к улучшению временной локальности использования данных.

Будем искать расписание θ как оптимальный набор аффинных отображений $\theta_{S_i}, i = 1, \ldots, m$ для E. Определим множество допустимых решений:

$$\begin{aligned}\Theta = \{\theta | (\theta(\langle \vec{x}, \vec{z}, S_i \rangle) = \vec{v}_{S_i} \cdot \vec{x} + \vec{v}'_{S_i} \cdot \vec{z} + v^0_{S_i},\\ v^0_{S_i} \in \mathbb{Z}, \vec{v}_{S_i} \in \mathbb{Z}^{p_{S_i}}, \vec{v}'_{S_i} \in \mathbb{Z}^{q_z}, \quad i = 1, \ldots, m) \wedge\\ (\forall \vec{x} \in D_{S_i} \colon \theta(\langle \vec{x}, \vec{z}, S_i \rangle) \geqslant 0, \quad i = 1, \ldots, m) \wedge\\ (\vec{y} \in P_{e_j} \Rightarrow \tau_{e_j}(\vec{y}, \vec{z}) \geqslant 1, \quad j = 1, \ldots, n)\}.\end{aligned}$$

Для каждого ребра e_j наложим ограничения на функции расписания инструкций $\sigma(e_j)$ и $\delta(e_j)$, затем составим систему из всех этих ограничений:

$$\begin{cases} \tau_{e_j}(\vec{y}, \vec{z}) - 1 \geqslant 0, \quad j = 1, \ldots, n \\ -\tau_{e_j}(\vec{y}, \vec{z}) + \vec{l}_{e_j} \cdot \vec{z} + l^0_{e_j} \geqslant 0, \quad j = 1, \ldots, n \end{cases}. \tag{2.6}$$

Первое ограничение обусловлено принципом причинности, второе задает лимит задержки использования данных, который требуется минимизировать.

Оптимальное расписание минимизирует целевую функцию (2.5) при ограничениях (2.6). Система ограничений (2.6) линеаризуется с помощью классической техники на основе леммы Фаркаша [27].

Лемма 1. (Лемма Фаркаша) *Пусть D — непустой многогранник, определённый с помощью p аффинных неравенств (граней): $\vec{a}_k \cdot \vec{x} + b_k \geqslant 0, k = 1, \ldots, p$. Аффинная функция ψ неотрицательна в любой точке $\vec{x} \in D$ тогда и только тогда, когда она является неотрицательной аффинной комбинацией: $\psi(\vec{x}) \equiv \lambda_0 + \sum_{k=1}^{p} \lambda_k (\vec{a}_k \cdot \vec{x} + b_k)$, $\lambda_* \geqslant 0$. Неотрицательные константы λ_* называются множителями Фаркаша.*

Для каждой инструкции S_i выписывается шаблон функции расписания, неотрицательной внутри ее домена:

$$\theta_{S_i}(\vec{x}_{S_i}, \vec{z}) = \mu_{S_i,0} + \sum_{k=1}^{p_{D_{S_i}}} \mu_{S_i,k} \left(\vec{a}_{S_i,k} \cdot \begin{bmatrix} \vec{x}_{S_i} \\ \vec{z} \end{bmatrix} + b_{S_i,k} \right), \quad i = 1, \ldots, m,$$

где $\mu_{S_i,*} \geqslant 0$ — множители Фаркаша, а $p_{D_{S_i}}$ неравенств $\vec{a}_{S_i,k} \cdot \begin{bmatrix} \vec{x}_{S_i} \\ \vec{z} \end{bmatrix} + b_{S_i,k} \geqslant 0$ определяют домен D_{S_i}.

Применим лемму Фаркаша для каждого ограничения. Пусть $\lambda_{e_j,*} \geqslant 0$ и $\lambda'_{e_j,*} \geqslant 0$ — множители Фаркаша, а $p_{P_{e_j}}$ неравенств $\vec{c}_{e_j,k} \cdot \begin{bmatrix} \vec{x}_{\delta(e_j)} \\ \vec{z} \end{bmatrix} + d_{e_j,k} \geqslant 0$

определяют многогранник зависимостей P_{e_j}:

$$\tau_{e_j}(\vec{y},\vec{z}) - 1 \equiv \lambda_{e_j,0} + \sum_{k=1}^{p_{P_{e_j}}} \lambda_{e_j,k} \left(\vec{c}_{e_j,k} \cdot \begin{bmatrix} \vec{x}_{\delta(e_j)} \\ \vec{z} \end{bmatrix} + d_{e_j,k} \right)$$

$$-\tau_{e_j}(\vec{y},\vec{z}) + \vec{l}_{e_j} \cdot \vec{z} + l^0_{e_j} \equiv \lambda'_{e_j,0} + \sum_{k=1}^{p_{P_{e_j}}} \lambda'_{e_j,k} \left(\vec{c}_{e_j,k} \cdot \begin{bmatrix} \vec{x}_{\delta(e_j)} \\ \vec{z} \end{bmatrix} + d_{e_j,k} \right)$$

В итоге система ограничений примет вид:

$$\begin{cases} \mu_{\delta(e_j),0} + \sum_{k=1}^{p_{D_{\delta(e_j)}}} \mu_{\delta(e_j),k} \left(\vec{a}_{\delta(e_j),k} \cdot \begin{bmatrix} \vec{x}_{\delta(e_j)} \\ \vec{z} \end{bmatrix} + b_{\delta(e_j),k} \right) - \\ \mu_{\sigma(e_j),0} - \sum_{k=1}^{p_{D_{\sigma(e_j)}}} \mu_{\sigma(e_j),k} \left(\vec{a}_{\sigma(e_j),k} \cdot \begin{bmatrix} h_e(\vec{x}_{\delta(e_j)}) \\ \vec{z} \end{bmatrix} + b_{\sigma(e_j),k} \right) - 1 = \\ \lambda_{e_j,0} + \sum_{k=1}^{p_{P_{e_j}}} \lambda_{e_j,k} \left(\vec{c}_{e_j,k} \cdot \begin{bmatrix} \vec{x}_{\delta(e_j)} \\ \vec{z} \end{bmatrix} + d_{e_j,k} \right), \quad j = 1,\ldots,n \\ \mu_{\sigma(e_j),0} + \sum_{k=1}^{p_{D_{\sigma(e_j)}}} \mu_{\sigma(e_j),k} \left(\vec{a}_{\sigma(e_j),k} \cdot \begin{bmatrix} h_e(\vec{x}_{\delta(e_j)}) \\ \vec{z} \end{bmatrix} + b_{\sigma(e_j),k} \right) - \\ \mu_{\delta(e_j),0} - \sum_{k=1}^{p_{D_{\delta(e_j)}}} \mu_{\delta(e_j),k} \left(\vec{a}_{\delta(e_j),k} \cdot \begin{bmatrix} \vec{x}_{\delta(e_j)} \\ \vec{z} \end{bmatrix} + b_{\delta(e_j),k} \right) + \vec{l}_{e_j} \cdot \vec{z} + l^0_{e_j} = \\ \lambda'_{e_j,0} + \sum_{k=1}^{p_{P_{e_j}}} \lambda'_{e_j,k} \left(\vec{c}_{e_j,k} \cdot \begin{bmatrix} \vec{x}_{\delta(e_j)} \\ \vec{z} \end{bmatrix} + d_{e_j,k} \right), \quad j = 1,\ldots,n \\ \mu_{S_{i,k}} \geqslant 0, \quad i = 1,\ldots,m, \quad k = 0,\ldots,p_{D_{S_i}} \\ \lambda_{e_{j,k}} \geqslant 0, \quad j = 1,\ldots,n, \quad k = 0,\ldots,p_{P_{e_j}} \\ \lambda'_{e_{j,k}} \geqslant 0, \quad j = 1,\ldots,n, \quad k = 0,\ldots,p_{P_{e_j}} \end{cases} . \quad (2.7)$$

Приравнивание соответственных множителей при $\vec{x}$ и $\vec{z}$ в левых и правых частях ограничений позволяет оставить в системе только множители Фаркаша μ, λ, λ' и переменные-ограничители $\vec{l}$ и l^0.

Поиск решения задачи оптимизации сводится к решению задачи линейного целочисленного программирования: минимизировать целевую функцию (2.5) при ограничениях (2.7). Решение задачи ЛЦП может быть осуществлено, например, методом ветвей и границ [104]. Подстановка значений μ в соответствующие шаблоны функций расписания для каждой инструкции дает решение задачи поиска расписания θ.

2.3 Многомерные аффинные расписания вычислений

Однако, не для всех линейных программ существует одномерное расписание. Футриером был предложен подход к распараллеливанию таких программ, заключающийся в поиске многомерных расписаний с аффинными компонентами. Жадный алгоритм, предложенный Футриером в работе [45], представляет практический интерес, поскольку в его основе лежит соображение о минимизации размерности расписания. Дополним жадный алгоритм Футриера новым этапом с целью выбора лучших одномерных аффинных компонентов многомерного расписания. Качество этих компонентов оценивается предложенной взвешенной суммой (2.5). Пусть множество $E^{'}$ — некоторое подмножество ребер обобщенного графа зависимостей, $E^{'} \subseteq E$; переменная d показывает индекс текущего компонента многомерного расписания, который должен быть вычислен. Получаем следующий жадный алгоритм поиска многомерных аффинных расписаний $\vec{\theta}$:

1. $E^{'} \leftarrow E$; $d \leftarrow 1$.
2. Построить систему:

$$\begin{cases} \tau_e(\vec{y}, \vec{z}) - \varepsilon_e \geqslant 0, e \in E^{'} \\ 0 \leqslant \varepsilon_e \leqslant 1, e \in E^{'} \end{cases} . \tag{2.8}$$

   Первое неравенство выражает принцип причинности для ребра e, если переменная-индикатор ε_e принимает значение 1. Алгоритм является жадным, так как пытается удовлетворить как можно больше зависимостей единовременно, решая задачу

$$\sum_{e \in E^{'}} \varepsilon_e \to \max \tag{2.9}$$

   при ограничениях (2.8). Задача решается методами линейного целочисленного программирования после линеаризации системы ограничений (классическая техника с применением леммы Фаркаша). Пусть $E_1^{'} = \{e | e \in E^{'} \wedge \varepsilon_e = 1\}$ — множество ребер, соответствующих удовлетворенным зависимостям, $E_0^{'} = \{e | e \in E^{'} \wedge \varepsilon_e = 0\}$ — множество ребер, соответствующих неудовлетворенным зависимостям.
3. Вычислить компонент $\vec{\theta}^{(d)}$ как одномерное аффинное расписание для подмножества ребер $E_1^{'} \subseteq E$, соответствующих удовлетворенным зависимостям.

4. Если $E_0^{'} = \varnothing$, процесс завершен и d показывает размерность многомерного расписания, иначе следует установить $E^{'} \leftarrow E_0^{'}$, $d \leftarrow d+1$ и перейти к шагу 2.

Шаг 3 расширяет классическую схему Футриера с тем, чтобы продолжить рассмотрение допустимых решений для каждого компонента многомерного расписания и выбрать наилучшее в соответствии с предложенными критериями качества.

Если для программы существует одномерное аффинное расписание, то оно будет найдено за одну итерацию алгоритма — все зависимости будут удовлетворены на шаге 2 при $d = 1$.

2.4 Одномерные аффинные размещения вычислений

Размещение обладает свойством вперед направленных коммуникаций (Forward Communication Only, FCO) тогда и только тогда, когда все направления коммуникаций содержатся в конусе $\left(0+, \ \ldots, \ 0+\right)$. Применение размещений, обладающих таким свойством, представляет практический интерес, поскольку позволяет в некоторых случаях вдвое уменьшить количество партнеров коммуникации. В работе М. Грибля [105] это свойство обсуждается подробно.

Пусть $\pi_{S_i}(\vec{x}, \vec{z}), \quad \vec{x} \in D_{S_i}$ — аффинная функция размещения для инструкции S_i. Рассмотрим функцию $\rho_{e_j}(\vec{y}, \vec{z}) = \pi_{\delta(e_j)}(\vec{y}, \vec{z}) - \pi_{\sigma(e_j)}(h_{e_j}(\vec{y}), \vec{z}), \quad \vec{y} \in P_{e_j}$. Эта функция дает меру расстояния использования данных. Минимизация этого расстояния приводит к улучшению пространственной локальности использования данных.

Предложенный Грибле метод предполагает четыре этапа:

1. Построение системы неравенств $\rho_{e_j}(\vec{y}, \vec{z}) \geqslant 0, j = 1, \ldots, n$, фиксирующих свойство вперед направленных коммуникаций.
2. Линеаризация системы ограничений с применением классической техники на основе леммы Фаркаша. Преобразованная система задает полиэдральный конус допустимых решений.
3. Поиск всех экстремальных лучей конуса с помощью алгоритма Черниковой [106; 107], реализованного в библиотеке PolyLib [108].

4. Выбор экстремального луча, соответствующего предпочтительному размещению, в соответствии с заданной целевой функцией (единственная вершина конуса соответствует нулевому решению, поэтому только экстремальные лучи представляют интерес).

Возможно выбрать оптимальное решение по этой схеме, применив целевую функцию (2.5). Однако, это влечет полный перебор экстремальных лучей и трудоемкую оценку качества соответствующих им размещений вычислений непосредственным применением утверждения 1. Предпочтительнее искать аффинное размещение вычислений π как оптимальный набор аффинных отображений $\pi_{S_i}, \quad i = 1, \ldots, m$ для E. Определим множество допустимых решений:

$$\begin{aligned}
\Pi = \{\pi | (\pi(\langle \vec{x}, \vec{z}, S_i \rangle) = \vec{v}_{S_i} \cdot \vec{x} + \vec{v}'_{S_i} \cdot \vec{z} + v^0_{S_i}, \\
v^0_{S_i} \in \mathbb{Z}, \vec{v}_{S_i} \in \mathbb{Z}^{p_{S_i}}, \vec{v}'_{S_i} \in \mathbb{Z}^{q_z}, \vec{v}_{S_i} \neq \vec{0}, \quad i = 1, \ldots, m) \wedge \\
(\forall \vec{x} \in D_{S_i} \colon \pi(\langle \vec{x}, \vec{z}, S_i \rangle) \geqslant 0, \quad i = 1, \ldots, m) \wedge \\
(\pi(\langle \vec{x}, \vec{z}, S_i \rangle) \neq \vec{\gamma}_{S_i} \cdot \vec{\theta}(\langle \vec{x}, \vec{z}, S_i \rangle) + \vec{\gamma}'_{S_i} \cdot \vec{z} + \gamma^0_{S_i}, \\
\gamma^0_{S_i} \in \mathbb{Z}, \vec{\gamma}_{S_i} \in \mathbb{Z}^d, \vec{\gamma}'_{S_i} \in \mathbb{Z}^{q_z}, \quad i = 1, \ldots, m) \wedge \\
(\vec{y} \in P_{e_j} \Rightarrow \rho_{e_j}(\vec{y}, \vec{z}) \geqslant 0, \quad j = 1, \ldots, n)\}.
\end{aligned}$$

d — размерность заранее вычисленного многомерного расписания.

Покажем, что функция размещения должна быть линейно независимой от компонентов многомерного расписания. Предположим, что это не выполняется, т.е. размещение представимо в виде $\pi(\langle \vec{x}, \vec{z}, S_i \rangle) = \vec{\gamma}_{S_i} \cdot \vec{\theta}(\langle \vec{x}, \vec{z}, S_i \rangle) + \vec{\gamma}'_{S_i} \cdot \vec{z} + \gamma^0_{S_i}, \gamma^0_{S_i} \in \mathbb{Z}, \vec{\gamma}_{S_i} \in \mathbb{Z}^d, \vec{\gamma}'_{S_i} \in \mathbb{Z}^{q_z}$. Рассмотрим две операции, принадлежащие одному фронту: $\langle \vec{x}_1, \vec{z}, S \rangle$ и $\langle \vec{x}_2, \vec{z}, S \rangle$, т.е. $\vec{\theta}(\langle \vec{x}_1, \vec{z}, S \rangle) = \vec{\theta}(\langle \vec{x}_2, \vec{z}, S \rangle)$. Из этого немедленно следует $\pi(\langle \vec{x}_1, \vec{z}, S \rangle) = \pi(\langle \vec{x}_2, \vec{z}, S \rangle)$. Получается, что размещение в этом случае не имеет параллелизма для любой отдельно взятой инструкции внутри любого фронта. Подобное имеет место и для нулевого решения — операции отдельной инструкции всегда попадают на один и тот же процессор. Недавно был предложен эффективный способ [109], позволяющий задать линейные ограничения, фиксирующие линейную независимость и исключающие из рассмотрения нулевое решение. Дальнейшие рассуждения опираются на следующий факт:

$$\begin{cases} c_i \in \mathbb{Z}, i = 1, \ldots, p \\ b \geqslant 2, b \in \mathbb{Z} \\ -b < c_i < b, i = 1, \ldots, p \\ \begin{bmatrix} c_1 & \ldots & c_p \end{bmatrix}^T \neq \vec{0} \end{cases} \Leftrightarrow \begin{cases} c_i \in \mathbb{Z}, i = 1, \ldots, p \\ b \geqslant 2, b \in \mathbb{Z} \\ -b < c_i < b, i = 1, \ldots, p \\ \varepsilon \in \{0, 1\} \\ \sum_{i=1}^{p} b^{i-1} c_i \geqslant 1 - \varepsilon b^{p+1} \\ -\sum_{i=1}^{p} b^{i-1} c_i \geqslant 1 - (1 - \varepsilon) b^{p+1} \end{cases} \tag{2.10}$$

Множество целочисленных векторов с ограниченными компонентами, не включающее нулевой вектор, можно описать системой линейных ограничений. Значит, возможно ограничить компоненты $\vec{v}_{S_i}$, выбрав параметр $b \geqslant 2$ и немедленно получить линейные ограничения, исключающие нулевое решение из рассмотрения. В работе [109] предлагается значение $b = 5$, как достаточное для большинства практических случаев.

Пусть $\boldsymbol{H}_{S_i}$ — матрица, строки которой представляют компоненты ранее найденного многомерного аффинного расписания для инструкции S_i:

$$\boldsymbol{H}_{S_i} = \begin{bmatrix} \vec{v}_{S_i,1}^T \\ \cdots \\ \vec{v}_{S_i,d}^T \end{bmatrix}.$$

Обозначим $\boldsymbol{H}_{S_i}^{\perp}$ подпространство ортогональное $\boldsymbol{H}_{S_i}$, т.е., любая строка $\boldsymbol{H}_{S_i}^{\perp}$ ортогональна любой строке $\boldsymbol{H}_{S_i}$ ($\boldsymbol{H}_{S_i}^{\perp} \boldsymbol{H}_{S_i}^T = \boldsymbol{0}$). Вычислить ортогональное подпространство можно следующим образом: $\boldsymbol{H}_{S_i}^{\perp} = \boldsymbol{I} - \boldsymbol{H}_{S_i}^T (\boldsymbol{H}_{S_i} \boldsymbol{H}_{S_i}^T)^{-1} \boldsymbol{H}_{S_i}$ [37]. Пусть $\boldsymbol{H}_{S_i}^{\perp}[r]$ — r-я строка матрицы $\boldsymbol{H}_{S_i}^{\perp}$, $r = 1, \ldots, p_{S_i}$. Для того, чтобы функция размещения $\pi(\langle \vec{x}, \vec{z}, S_i \rangle)$ была линейно независимой от компонентов многомерного аффинного расписания, $\vec{v}_{S_i}$ должен иметь хотя бы один ненулевой компонент в ортогональном подпространстве $\boldsymbol{H}_{S_i}^{\perp}$ [37]:

$$\begin{bmatrix} \boldsymbol{H}_{S_i}^{\perp}[1] \vec{v}_{S_i} & \ldots & \boldsymbol{H}_{S_i}^{\perp}[p_{S_i}] \vec{v}_{S_i} \end{bmatrix}^T \neq \vec{0}. \tag{2.11}$$

Если установлены ограничения на компоненты $\vec{v}_{S_i}$ с некоторым параметром $b \geqslant 2$, то компоненты вектора в условии (2.11) являются ограниченными:

$$\exists B \in \mathbb{Z} \wedge B \geqslant 2 \left(-B < \boldsymbol{H}_{S_i}^{\perp}[r] \vec{v}_{S_i} < B, r = 1, \ldots, p_{S_i} \right).$$

Аналогичным образом применяется (2.10) к условию (2.11) для получения ограничений линейной независимости. Параметр B выбирается следующим образом:

$$B = \max_{r=1,\ldots,p_{S_i}} \{ | \max_{-b<\vec{v}_{S_i}[l]<b, l=1,\ldots,p_{S_i}} (\boldsymbol{H}_{S_i}^{\perp}[r]\vec{v}_{S_i})|, \\ | \min_{-b<\vec{v}_{S_i}[l]<b, l=1,\ldots,p_{S_i}} (\boldsymbol{H}_{S_i}^{\perp}[r]\vec{v}_{S_i})| \} + 1. \tag{2.12}$$

Поиск экстремумов здесь может быть выполнен методами линейного целочисленного программирования.

Комбинируя все ограничения на функции размещения вычислений в единую систему, получим:

$$\begin{cases} \rho_{e_j}(\vec{y},\vec{z}) \geqslant 0, \quad j = 1,\ldots,n \\ -\rho_{e_j}(\vec{y},\vec{z}) + \vec{l}_{e_j} \cdot \vec{z} + l_{e_j}^0 \geqslant 0, \quad j = 1,\ldots,n \\ \vec{v}_{S_i} \neq \vec{0}, \quad i = 1,\ldots,m \\ \left[\boldsymbol{H}_{S_i}^{\perp}[1]\vec{v}_{S_i} \quad \ldots \quad \boldsymbol{H}_{S_i}^{\perp}[p_{S_i}]\vec{v}_{S_i} \right]^T \neq \vec{0}, \quad i = 1,\ldots,m. \end{cases} \tag{2.13}$$

Первое ограничение обеспечивает свойство вперед направленных коммуникаций, второе задает лимит расстояния использования данных, который требуется минимизировать, третье исключает из рассмотрения нулевое решение, четвертое обеспечивает линейную независимость от компонентов многомерного аффинного расписания.

Оптимальное размещение вычислений минимизирует (2.5) при ограничениях (2.13). Задача решается методами линейного целочисленного программирования после линеаризации первого и второго ограничений в системе (2.13) применением классической техники на основе леммы Фаркаша. Подстановка значений μ в соответствующие шаблоны функций размещения для каждой инструкции дает решение задачи поиска размещения π.

Рассмотрение лишь одномерных аффинных размещений не означает серьезного сужения практической применимости. Множество вычислительных ядер однородной кластерной системы успешно моделируется одномерным пространством виртуальных процессоров в терминах модели многогранников, как и набор ядер многоядерного процессора. Одномерная вычислительная сетка в терминах спецификаций OpenCL и CUDA успешно моделируется таким же образом при взаимно однозначном соответствии между вычислительной нитью и виртуальным процессором.

2.5 Одномерные аффинные размещения вычислений и данных. Случай явно заданного расположения данных

Пусть $\eta_A(\vec{g},\vec{z}), \quad \vec{g} \in D_A$ — аффинная функция размещения для массива A. Рассмотрим функцию

$$\nu_a(\vec{x},\vec{z}) = \begin{cases} \pi_{S_a}(\vec{x},\vec{z}) - \eta_{A_a}(g_a(\vec{x}),\vec{z}), \vec{x} \in D_{S_a} & \text{если } a \text{ — чтение} \\ \eta_{A_a}(g_a(\vec{x}),\vec{z}) - \pi_{S_a}(\vec{x},\vec{z}), \vec{x} \in D_{S_a} & \text{если } a \text{ — запись} \end{cases}.$$

Функция ν_a дает меру расстояния использования данных d_a^{ρ}. Минимизация этого расстояния снижает издержки межпроцессорной коммуникации.

Ответ на вопрос о верхней границе для d_a^{ρ} дает следующее утверждение.

Утверждение 2. *Если домен D_{S_a} является ограниченным, и заданы два аффинных отображения $\pi_{S_a}(\vec{x},\vec{z})$ и $\eta_{A_a}(g_a(\vec{x}),\vec{z})$, то найдется аффинная функция $L(\vec{z})$, зависящая только от внешних переменных программы, такая, что*

$$L(\vec{z}) - \nu_a(\vec{x},\vec{z}) \geqslant 0, \quad x_{S_a} \in D_{S_a}.$$

Доказательство. Поскольку индексная функция доступа g_a является аффинной, то ее можно задать матрицей $\boldsymbol{G}_a$:

$$g_a(\vec{x}) = \boldsymbol{G}_a \begin{bmatrix} \vec{x} \\ \vec{z} \\ 1 \end{bmatrix}, \quad \boldsymbol{G}_a \in \mathbb{Q}^{p_{A_a} \times (p_{S_a} + q_z + 1)}$$

Разобьем матрицу $\boldsymbol{G}_a$ на блоки, отдельно участвующие в умножении на $\vec{x}$, $\vec{z}$, 1:

$$g_a(\vec{x}) = \begin{bmatrix} \boldsymbol{G}_a^{\vec{x}} & \boldsymbol{G}_a^{\vec{z}} & \boldsymbol{G}_a^1 \end{bmatrix} \begin{bmatrix} \vec{x} \\ \vec{z} \\ 1 \end{bmatrix} = \boldsymbol{G}_a^{\vec{x}}\vec{x} + \boldsymbol{G}_a^{\vec{z}}\vec{z} + \boldsymbol{G}_a^1$$

Очевидно, что $\nu_a(\vec{x},\vec{z}) = |\vec{v}_{S_a} \cdot \vec{x} - \vec{v}_{A_a} \cdot (\boldsymbol{G}_a^{\vec{x}}\vec{x} + \boldsymbol{G}_a^{\vec{z}}\vec{z} + \boldsymbol{G}_a^1) + (\vec{v}'_{S_a} - \vec{v}'_{A_a}) \cdot \vec{z} + v^0_{S_a} - v^0_{A_a}| = |(\vec{v}^T_{S_a} - \vec{v}^T_{A_a}\boldsymbol{G}_a^{\vec{x}})\vec{x} + (\vec{v}'^T_{S_a} - \vec{v}'^T_{A_a} - \vec{v}^T_{A_a}\boldsymbol{G}_a^{\vec{z}})\vec{z} + v^0_{S_a} - v^0_{A_a} - \vec{v}^T_{A_a}\boldsymbol{G}_a^1|$.

D_{S_a} — ограниченный выпуклый многогранник, заданный параметрически, значит, существует решение задачи линейного программирования $\nu_a(\vec{x},\vec{z}) \to \max\limits_{\vec{x} \in D_{S_a}}$. Максимум $\nu_a(\vec{x},\vec{z})$ достигается в одной из вершин D_{S_a}.

Внешние переменные программы $\vec{z}$ линейно входят в уравнения гиперплоскостей, определяющих грани D_{S_a}. Поскольку вершины D_{S_a} являются точками пересечения нескольких граней, то их координаты задаются аффинными функциями, зависящими от $\vec{z}$. Следовательно, значение ν_a в любой вершине представимо аффинной функцией, зависящей от $\vec{z}$. Для каждой вершины S с координатами $\vec{s}$ имеем:

$$\exists \boldsymbol{V}_S \in \mathbb{Q}^{p_{S_a} \times q_z}, \vec{v}_S \in \mathbb{Q}^{p_{S_a}}, \vec{u}_S \in \mathbb{Q}^{q_z}, u_S^0 \in \mathbb{Q}(\vec{s} = \boldsymbol{V}_S \vec{z} + \vec{v}_S \Rightarrow \nu_a(\vec{s}, \vec{z}) = \vec{u}_S \cdot \vec{z} + u_S^0).$$

Перебрав все вершины S ограниченного многогранника D_{S_a}, можно выбрать любое $L(\vec{z}) = \vec{l} \cdot \vec{z} + l^0$, удовлетворяющее условию:

$$l^0 \geqslant \max_S(u_S^0), \quad \vec{l}[i] \geqslant \max_S(\vec{u}_S[i]), i = 1 \ldots q_z \tag{2.14}$$

□

Из утверждения 2 следует, что можно выбрать аффинную функцию $L_e(\vec{z}) = \vec{l_e} \cdot \vec{z} + l_e^0$ с наименьшими коэффициентами, ограничивающую сверху $\nu_a(\vec{x}, \vec{z})$. Это можно сделать, рассмотрев ограничения в условии (2.14) как равенства.

Определение Пусть $\left\{\varphi \middle| \varphi = \left\langle \bigcup\limits_{a \in \{a\}} \{\pi_{S_a}\}, \bigcup\limits_{a \in \{a\}} \{\eta_{A_a}\} \right\rangle \right\}_{\{a\}}$ — множество допустимых наборов φ одномерных аффинных отображений π_{S_a} и η_{A_a} для всех инструкций S_a и массивов A_a в программе, порождаемое множеством $\{a\}$ доступов к данным a. Набор φ называется оптимальным для программы, если он минимизирует функционалы $f_e(\varphi) = L_a(\vec{z}) = \vec{l_a} \cdot \vec{z} + l_a^0$, для всех доступов к данным a так, что

$$\nexists \varphi' \in \{\varphi\}_{\{a\}} \left(\forall a \in \{a\} \left(f_a(\varphi') \leqslant f_a(\varphi) \right) \wedge \exists a \in \{a\} \left(f_e(\varphi') < f_e(\varphi) \right) \right).$$

Нахождение оптимального набора подразумевает решение задачи оптимизации, где:

1. $\{\varphi\}_{\{a\}}$ — множество допустимых решений.
2. $f_a(\varphi) = L_a(\vec{z})$, $a \in \{a\}$ — показатели качества решения φ (меньшие значения свидетельствуют о более предпочтительном решении).

Будем искать решение, применив взвешенную сумму:

$$\sum_{a \in \{a\}} \alpha_a f_a \to \min, \quad \alpha_a > 0,$$

где α_a — весовые коэффициенты, рассматриваемые как показатели относительной значимости доступов к данным.

Оптимальное решение минимизирует

$$C^{\nu}_{\{a\}}(\varphi) = \sum_{a \in \{a\}} \alpha_a (\vec{l_a} \cdot \vec{z} + l_a^0). \tag{2.15}$$

С одной стороны, оптимальное решение не может быть улучшено ни по какому из показателей качества $L_a(\vec{z}) = \vec{l_a} \cdot \vec{z} + l_a^0$. С другой стороны, каждое полученное $L_a(\vec{z})$ ограничивает сверху $\nu_a(\vec{x}, \vec{z}), \quad x \in D_{S_a}$. Значит, $L_a(\vec{z}) = \max\limits_{\vec{x} \in D_{S_a}} \nu_a(\vec{x}, \vec{z})$. Следовательно, оптимальный набор аффинных отображений доставит минимум (2.3).

Такая формулировка в силу большей гибкости относительно [37; 66] позволяет описать естественную эвристику: улучшение локальности использования данных более значимо для тех доступов к данным, которые чаще осуществляются. Для программ со слабой параметризацией, или в случае JIT-трансляции, когда значения внешних переменных программы становятся известными, становится возможной приоритизация доступов к данным на основе частоты их возникновения: $\alpha_a = \#D_{S_a}$, где $\#D_{S_a}$ — количество точек с целочисленными координатами внутри домена, которое может быть вычислено с помощью алгоритма Клаусса [103]. В случае статической компиляции рекомендуется выбрать значения $\vec{z}$ как можно более близкие к вероятным исходя из априорной информации относительно контекста, а затем вычислить $\#D_{S_a}$ в соответствии с предполагаемыми значениями.

Далее формулировка задачи для отыскания размещений вычислений и данных будет уточняться путем задания набора ограничений. Зафиксируем $k = |\{a\}|$ — количество обращений a к массивам в программе, l — количество массивов, к которым выполняется доступ.

Предложенный Грибльм метод предполагает четыре этапа:

1. Построение системы неравенств $\nu_{a_j}(\vec{x}, \vec{z}) \geqslant 0, j = 1, \ldots, k$, фиксирующих свойство вперед направленных коммуникаций.
2. Линеаризация системы ограничений с применением классической техники на основе леммы Фаркаша. Преобразованная система задает полиэдральный конус допустимых решений.
3. Поиск всех экстремальных лучей конуса с помощью алгоритма Черниковой [106; 107], реализованного в библиотеке PolyLib [108].

4. Выбор экстремального луча, соответствующего предпочтительному размещению, в соответствии с заданной целевой функцией (единственная вершина конуса соответствует нулевому решению, поэтому только экстремальные лучи представляют интерес).

Возможно выбрать оптимальное решение по этой схеме, применив целевую функцию (2.15). Однако, это влечет полный перебор экстремальных лучей и трудоемкую оценку качества соответствующих им размещений вычислений непосредственным применением утверждения 2. Предпочтительнее искать оптимальный набор аффинных отображений $\langle\{\pi_{S_i}, i = 1, \ldots, m\}, \{\eta_{A_j}, j = 1, \ldots l\}\rangle$. Определим множество допустимых решений:

$$\{\langle\pi, \eta\rangle | (\pi \in \Pi, \eta \in H)\},$$

где

$$\begin{aligned} H = \{\eta | (\eta(\langle\vec{g}, \vec{z}, A_j\rangle) = \vec{v}_{A_j} \cdot \vec{g} + \vec{v}'_{A_j} \cdot \vec{z} + v^0_{A_j}, \\ v^0_{A_j} \in \mathbb{Z}, \vec{v}_{A_j} \in \mathbb{Z}^{p_{A_j}}, \vec{v}'_{A_j} \in \mathbb{Z}^{q_z}, \quad j = 1, \ldots, l) \wedge \\ (\forall \vec{g} \in D_{A_j} : \eta(\langle\vec{g}, \vec{z}, A_j\rangle) \geqslant 0, \quad j = 1, \ldots, l)\}. \end{aligned}$$

Функция размещения вычислений должна быть линейно независимой от компонентов многомерного расписания. Скомбинируем все ограничения на функции размещения вычислений и данных в одну систему:

$$\begin{cases} \nu_{a_j}(\vec{x}, \vec{z}) \underset{\vec{x} \in D_{S_{a_j}}}{\geqslant} 0, \quad j = 1, \ldots, k \\ -\nu_{a_j}(\vec{x}, \vec{z}) + \vec{l}_{a_j} \cdot \vec{z} + l^0_{a_j} \underset{\vec{x} \in D_{S_{a_j}}}{\geqslant} 0, \quad j = 1, \ldots, k \\ \vec{v}_{A_{a_j}} \cdot \vec{g} + \vec{v}'_{A_{a_j}} \cdot \vec{z} + v^0_{A_{a_j}} \underset{\vec{g} \in G_{a_j}}{\geqslant} 0, \quad j = 1, \ldots, k \\ \vec{v}_{S_i} \neq \vec{0}, \quad i = 1, \ldots, m \\ \left[\boldsymbol{H}^{\perp}_{S_i}[1]\vec{v}_{S_i} \quad \ldots \quad \boldsymbol{H}^{\perp}_{S_i}[p_{S_i}]\vec{v}_{S_i} \right]^T \neq \vec{0}, \quad i = 1, \ldots, m \end{cases}, \tag{2.16}$$

где G_{a_j} — образ домена $D_{S_{a_j}}$ по индексной функции доступа g_{a_j}. Точки с целочисленными координатами, принадлежащие многограннику G_{a_j}, формируют множество индексов массива A_{a_j}, к которому выполняется доступ в инструкции S_{a_j}.

Первое ограничение обеспечивает свойство вперед направленных коммуникаций, второе задает лимит расстояния использования данных, который

требуется минимизировать, третье позволяет вычислить общий вид функций размещения данных с помощью метода неопределенных коэффициентов, четвертое исключает из рассмотрения нулевое решение, пятое обеспечивает линейную независимость от компонентов многомерного аффинного расписания.

Оптимальное размещение вычислений и данных минимизирует целевую функцию (2.15) при ограничениях (2.16). Задача решается методами линейного целочисленного программирования после линеаризации первого, второго и третьего ограничений в системе (2.16) применением классической техники на основе леммы Фаркаша.

Для каждой инструкции S_i выписывается шаблон функции размещения вычислений, неотрицательной внутри ее домена:

$$\pi_{S_i}(\vec{x}_{S_i}, \vec{z}) = \mu_{S_i,0} + \sum_{k=1}^{p_{D_{S_i}}} \mu_{S_i,k} \left(\vec{a}_{S_i,k} \cdot \begin{bmatrix} \vec{x}_{S_i} \\ \vec{z} \end{bmatrix} + b_{S_i,k} \right), \quad i = 1, \ldots, m, \qquad (2.17)$$

где $\mu_{S_i,*} \geqslant 0$ — множители Фаркаша, а $p_{D_{S_i}}$ неравенств $\vec{a}_{S_i,k} \cdot \begin{bmatrix} \vec{x}_{S_i} \\ \vec{z} \end{bmatrix} + b_{S_i,k} \geqslant 0$ определяют домен D_{S_i}.

Для каждого обращения a_j выписывается шаблон функции размещения данных, неотрицательной внутри G_{a_j}:

$$\eta_{A_{a_j}}(\vec{g}_{a_j}, \vec{z}) = \mu_{a_j,0} + \sum_{k=1}^{p_{G_{a_j}}} \mu_{a_j,k} \left(\vec{a}_{a_j,k} \cdot \begin{bmatrix} \vec{g}_{a_j} \\ \vec{z} \end{bmatrix} + b_{a_j,k} \right), \quad j = 1, \ldots, k,$$

где $\mu_{a_j,*} \geqslant 0$ — множители Фаркаша, а $p_{G_{a_j}}$ неравенств $\vec{a}_{a_j,k} \cdot \begin{bmatrix} \vec{g}_{a_j} \\ \vec{z} \end{bmatrix} + b_{a_j,k} \geqslant 0$ определяют образ G_{a_j}. Подставляя $g_{a_j}(\vec{i}_{S_{a_j}})$ вместо $\vec{g}_{a_j}$ получаем размещение данных, зависящее от индекса итерации инструкции S_{a_j}:

$$\eta_{A_{a_j}}(\vec{i}_{S_{a_j}}, \vec{z}) = \mu_{a_j,0} + \sum_{k=1}^{p_{G_{a_j}}} \mu_{a_j,k} \left(\vec{a}_{a_j,k} \cdot \begin{bmatrix} g_{a_j}(\vec{i}_{S_{a_j}}) \\ \vec{z} \end{bmatrix} + b_{a_j,k} \right), \quad j = 1, \ldots, k. \qquad (2.18)$$

Первое и второе ограничения в системе (2.16) линеаризуются с помощью выражений (2.17) и (2.18), а третье — с помощью (2.18) после подстановки $g_{a_j}(\vec{i}_{S_{a_j}})$ вместо $\vec{g}_{a_j}$.

Подстановка значений μ в соответствующие шаблоны функций размещения вычислений для каждой инструкции дает решение задачи поиска размещения π, а совместно вычисляемое размещение данных η получается непосредственно благодаря третьему ограничению в (2.16).

2.6 Пример: распараллеливание LU-разложения

Рассмотрим реализацию алгоритма LU-разложения квадратной матрицы A из пакета Polybench [110], приведенную на листинге 2.1. Детальное описание алгоритма и обобщенный граф зависимостей приведены в приложении Б.

Листинг 2.1 LU-разложение из пакета Polybench на языке C

```
for (int k = 0; k < N; k++) {
  for (int l = k + 1; l < N; l++)
    A[l][k] /= A[k][k]; //S1
  for (int i = k + 1; i < N; i++)
    for (int j = k + 1; j < N; j++)
      A[i][j] -= A[i][k] * A[k][j]; //S2
}
```

Сравним две его параллельные реализации, полученные с помощью разработанного метода. Для обозначения параллелизма циклов используются директивы стандарта OpenMP. Для генерации параллельной программы используется библиотека cloog [88] версии 0.18.4. Используются директивы препроцессора языка C, приведенные на листинге 2.2.

Листинг 2.2 Директивы препроцессора для LU-разложения

```
#define min(x, y) (((x) < (y)) ? (x) : (y))
#define max(x, y) (((x) > (y)) ? (x) : (y))
#define S1(k,l) A[l][k] /= A[k][k]
#define S2(k,i,j) A[i][j] -= A[i][k] * A[k][j]
#define ceild(n,d)  ceil(((double)(n))/((double)(d)))
#define floord(n,d) floor(((double)(n))/((double)(d)))
```

Из описания реализации LU-разложения известно, что параметр `N` задает размер задачи (порядок квадратной матрицы A) и должен быть большим положительным числом, поэтому целесообразно выбрать $\vec{z} = [1000]$. Распараллеливание с применением разработанного метода с соблюдением всех рекомендаций дает $\theta_{S_1} = 2k$, $\theta_{S_2} = 2k + 1$, $\pi_{S_1} = l - 1$, $\pi_{S_2} = i - 1$ (результирующая параллельная программа представлена на листинге 2.3). В случае, если ничего неизвестно о контексте, выбирается $\vec{z} = [1]$ и $\alpha_{e_j} = 1$ для всех e_j. Это дает $\theta_{S_1} = 2k$, $\theta_{S_2} = k + j$, $\pi_{S_1} = l - 1$, $\pi_{S_2} = i - 1$ (результирующая параллельная программа представлена на листинге 2.4).

Эксперименты проводились для матрицы размера 2048×2048 вещественных чисел двойной точности на восьмиядерном процессоре Intel Xeon E2690 под

Листинг 2.3 Параллельный вариант LU-разложения на языке C (контекст известен)

```
#pragma omp parallel
{
  for (int p1=0;p1<=2*N-3;p1++) {
    #pragma omp for
    for (int p2=ceild(p1-1,2);p2<=N-2;p2++) {
      if (p1%2 == 0)
        S1((p1/2),(p2+1));
      if ((p1+1)%2 == 0)
        for (int j=ceild(p1+1,2);j<=N-1;j++)
          S2(((p1-1)/2),(p2+1),j);
    }
  }
}
```

Листинг 2.4 Параллельный вариант LU-разложения на языке C (контекст неизвестен)

```
#pragma omp parallel
{
  for (int p1=0;p1<=2*N-3;p1++) {
    #pragma omp for
    for (int p2=max(0,p1-N+1);p2<=N-2;p2++) {
      if (p1 <= 2*p2)
        if (p1%2 == 0)
          S1((p1/2),(p2+1));
      for (int k=max(0,p1-N+1);k<=min(floord(p1-1,2),p2);k++)
        S2(k,(p2+1),(p1-k));
    }
  }
}
```

операционной системой Linux CentOS 6.5 x64. Компиляция производилась с помощью gcc версии 4.4.7 с флагом оптимизации -O3. Ускорение, достигаемое при параллельном выполнения LU-разложения в 8 нитей относительно изначального последовательного варианта, составило: 7,31 раза для варианта на листинге 2.3 и 0,94 раза для варианта на листинге 2.4 (кэш-промахи негативно сказываются на производительности).

Даже в случаях, когда ничего не известно о контексте, не рекомендуется устанавливать вес, равный 1, для переменных $\vec{l}_e$. Это задает одинаковый приоритет при оптимизации для $\vec{l}_e$ и l_e^0, что противоречит соображению о предпочтительности ограничения задержек и расстояний использования данных постоянным числом. Также не рекомендуется рассмотрение всех зависимостей как равновесных. Всегда можно выбрать положительные значения большие 1 для компонентов

$\vec{z}$, а также размерности многогранников зависимостей P_{e_j} для весов α_{e_j}. Применение подобных ослабленных рекомендаций в рассматриваемом случае позволяет получить такой же результат, что и с $\alpha_{e_j} = \#P_{e_j}$.

Применение разработанного метода для вычисления размещения вычислений и данных совместно дает следующие отображения: $\pi_{S_1} = l$, $\pi_{S_2} = i$, $\eta_A = \vec{g}_A^{(0)}$. Результирующая параллельная программа, сгенерированная cloog [88], представлена на листинге 2.5 с дополнительными аннотациями в комментариях: обозначениями инструкций S и доступов к данным r и w для чтения и записи соответственно.

Листинг 2.5 Параллельный вариант LU-разложения на языке C (контекст известен)

```
1  for (int c1=0; c1<=2*N-3; c1++) {
2    for (int c2=ceild(c1+1,2); c2<=N-1; c2++) {
3      if ((c1+1)%2==0)
4        for (int c8=ceild(c1+1,2); c8<=N-1; c8++)
5          //S1 r1,w1          r2           r3
6          A[c2][c8]-=A[c2][(c1-1)/2]*A[(c1-1)/2][c8];
7      if (c1%2==0)
8        //S2 r4,w2       r5
9        A[c2][c1/2]/=A[c1/2][c1/2];
10   }
11 }
```

Цикл по `c2` является параллельным, поскольку итерирует по пространству виртуальных процессоров, а его тело содержит операции яруса параллельной формы. Из всех обращений к массивам только r_3 и r_5 предполагают удаленный доступ к данным (на чтение), остальные обращения к массивам оперируют строкой матрицы `A[c2]`, расположенной на том же виртуальном процессоре, что и выполняющаяся операция: в рассматриваемом случае это виртуальный процессор с индексом `c2`. Технические детали распределения операций яруса параллельной формы между физическими процессорами, а также организация информационного обмена между параллельными процессами для реализации операций удаленного доступа обсуждаются в главе 3.

2.7 Общие выводы по главе

В ходе исследования была показана необходимость совершенствования критериев оптимальности пространственных и временных отображений программ линейного класса, позволяющих количественно описать локальность использования данных. Предложен метод поиска оптимальных отображений, нацеленный на улучшение временной и пространственной локальности использования данных и, следовательно, снижение издержек межпроцессорной коммуникации. Рассмотрение задач поиска расписаний вычислений и размещений вычислений и данных с предложенной взвешенной суммой показателей качества решения позволяет находить более удачные (в плане повышения быстродействия синтезированной параллельной программы) пространственные и временные отображения, чем методы, опирающиеся на технику поиска лексикографического минимума на многограннике. Наилучшие результаты могут быть достигнуты для программ со слабой параметризацией или программ с искусственно ослабленной параметризацией во время JIT-трансляции. Рекомендуется применение разработанного метода в средах, применяющих JIT-подход, для осуществления распараллеливания во время выполнения программы. При статической компиляции должна использоваться вся информация о контексте для наиболее точной оценки внешних переменных программы. Вопрос оптимальности многомерного аффинного отображения, включающего измерения расписания и размещения вычислений, остается открытым и является предметом дальнейших исследований.

Глава 3. Инструментальная поддержка распараллеливания программ линейного класса

3.1 Организация информационного обмена между параллельными процессами

Рассматривается параллельная программа, сгенерированная cloog [88]. Расписание вычислений, а также размещение вычислений и данных, получены применением метода, описанного в главе 2, то есть предполагается одномерное пространство виртуальных процессоров. Предположим, что параллельная программа содержит l массивов A_k, $k = 1, \ldots, l$, m инструкций S_i, $i = 1, \ldots, m$, n обращений к памяти a_j, $j = 1, \ldots, n$.

На практике часто применяется равномерное распределение виртуальных процессоров между физическими («блочная» схема распределения процессоров [67]). Предположим, что вычислительная система имеет Q физических процессоров. Проиндексируем их значениями $0, \ldots, Q-1$ и введем переменную-индекс r. Тогда каждый физический процессор r обрабатывает отрезок пространства виртуальных процессоров $l(r), \ldots, u(r)$, как определено в (3.1).

$$\begin{aligned} l(r+1) &= u(r) + 1, \\ L \leqslant l(r) &\leqslant u(r) \leqslant U, \\ l(r) &= L + r \left\lfloor \frac{U - L + Q}{Q} \right\rfloor, \\ L &= \min(\min_{i=1,\ldots,m}(\pi_{S_i}), \min_{k=1,\ldots,l}(\eta_{A_k})), \\ U &= \max(\max_{i=1,\ldots,m}(\pi_{S_i}), \max_{k=1,\ldots,l}(\eta_{A_k})). \end{aligned} \tag{3.1}$$

В случае, когда $U - L + 1 < Q$, не все Q физических процессоров могут быть нагружены.

Обмен данными между физическими процессорами необходим для реализации операций удаленного чтения и удаленной записи. Пусть процессор с индексом R осуществляет доступ к элементу данных, расположенному на процессоре с индексом r. Если размещение вычислений и данных имеет свойство вперед направленных коммуникаций, то для операций удаленного чтения выполняется $R > r$, а для операций удаленной записи — $R < r$.

Выполним построение программы с синхронным параллелизмом по следующей схеме:

```
1 do t = 0; L
2   for remote reads exchange data
3   pardo F(t)
4   barrier
5   for remote writes exchange data
6 end do
```

Перед выполнением яруса параллельной формы происходит информационный обмен между физическими процессорами для реализации операций удаленного чтения, а после — информационный обмен с участием результатов, полученных при выполнении яруса, для реализации операций удаленной записи. Обмен данными реализуется в рамках стандарта MPI, коммуникация процессов — двухсторонняя (применяются функции `MPI_Send`, `MPI_Recv`, также подходят и их неблокирующие варианты). Для применяемой в работе блочной схемы распределения процессоров определим участки массивов, которые должны быть переданы или приняты в рамках реализации операций удаленного доступа.

Обозначим $D_S^{'}$ — домен инструкции S в параллельной программе, $p_S^{'}$ — его размерность, $\vec{i}^{'}_S \in D_S^{'}$ — целочисленный вектор индекса итерации, $\vec{i}^{'(p)}_S$ — индекс цикла, итерирующего по пространству виртуальных процессоров.

Пусть C — многогранник, описывающий внешние переменные программы (далее — «контекст» [88]), заданный p_C ограничениями вида $\vec{c}_k \cdot \vec{z} + d_k \geqslant 0$, $\vec{c}_k \in \mathbb{Z}^{q_z}$, $d_k \in \mathbb{Z}$, $k = 1, \ldots, p_C$.

Пусть $D_S^{'}$ задан $p_{D_S^{'}}$ ограничениями вида $\vec{b}_{S,k} \cdot \vec{i}^{'}_S + \vec{c}_{S,k} \cdot \vec{z} + d_{S,k} \geqslant 0$, $\vec{b}_{S,k} \in \mathbb{Z}^{p_S^{'}}$, $\vec{c}_{S,k} \in \mathbb{Z}^{q_z}$, $d_{S,k} \in \mathbb{Z}$, $k = 1, \ldots, p_{D_S^{'}}$.

Определение Параметрически заданный многогранник $D^{''}_{S,R}$ называется мгновенным доменом инструкции S для физического процессора R:

$$\vec{i}^{''}_S = \begin{bmatrix} \vec{i}^{'(p)}_S \\ \vdots \\ \vec{i}^{'(p_S^{'})}_S \end{bmatrix} \in D^{''}_{S,R} \Leftrightarrow \begin{cases} l(R) \leqslant \vec{i}^{'(p)}_S \leqslant u(R) \\ \vec{b}_{S,k} \cdot \vec{i}^{'}_S + \vec{c}_{S,k} \cdot \vec{z} + d_{S,k} \geqslant 0, k \in K_S^{D^{''}} \end{cases},$$

где $K_S^{D^{''}} = \{k | k = 1, \ldots, p_{D_S^{'}} \wedge \sum\limits_{l=p+1}^{p_S^{'}} |\vec{b}^{(l)}_{S,k}| > 0\}$ — множество индексов ограничений, включающих хотя бы один из компонентов $\vec{i}^{'(l)}_S$, $l = p+1, \ldots, p_S^{'}$.

Контекст $C''_{S,R}$ определяет параметры $D''_{S,R}$ и задается ограничениями:

$$\begin{cases} \vec{c}_k \cdot \vec{z} + d_k \geqslant 0, k = 1, \ldots, p_C \\ \vec{b}_{S,k} \cdot \vec{i'}_S + \vec{c}_{S,k} \cdot \vec{z} + d_{S,k} \geqslant 0, k \in K_S^{C''} \end{cases},$$

где $K_S^{C''} = \{k | k = 1, \ldots, p_{D'_S} \wedge \sum\limits_{l=p+1}^{p'_S} |\vec{b}_{S,k}^{(l)}| = 0\}$ — множество индексов ограничений, не включающих ни один из компонентов $\vec{i'}_S^{(l)}$, $l = p+1, \ldots, p'_S$.

Пусть $\widetilde{g}_a(\theta', D''_{S,R})$ — множество индексов элементов массива A_a, к которым осуществляется доступ a в некоторой позиции инструкции S физическим процессором R в момент времени $\theta' = \begin{bmatrix} \vec{i'}_S^{(1)} \\ \vdots \\ \vec{i'}_S^{(p-1)} \end{bmatrix}$.

Участок массива A_a, который должен быть принят процессором R от процессора r перед выполнением яруса параллельной формы для реализации удаленного чтения a в некоторой позиции инструкции S, определяется следующими ограничениями на индексы массива $\vec{g}$:

$$Q_a^{r-r}(\vec{g}, R, r) : \begin{cases} \vec{g} \in \widetilde{g}_a(\theta', D''_{S,R}) \\ l(r) \leqslant \eta_{A_a}(\vec{g}, \vec{z}) \leqslant u(r) \end{cases}.$$

Участок массива A_a, который должен быть передан процессору r процессором R перед выполнением яруса параллельной формы для реализации удаленного чтения a в некоторой позиции инструкции S, определяется следующими ограничениями на индексы массива $\vec{g}$:

$$Q_a^{r-s}(\vec{g}, R, r) : \begin{cases} \vec{g} \in \widetilde{g}_a(\theta', D''_{S,r}) \\ l(R) \leqslant \eta_{A_a}(\vec{g}, \vec{z}) \leqslant u(R) \end{cases}.$$

Участок массива A_a, который должен быть передан процессору r процессором R после выполнения яруса параллельной формы для реализации удаленной записи a в некоторой позиции инструкции S, определяется следующими ограничениями на индексы массива $\vec{g}$:

$$Q_a^{w-s}(\vec{g},R,r):\begin{cases}\vec{g}\in\widetilde{g}_a(\theta',D''_{S,R})\\ l(r)\leqslant\eta_{A_a}(\vec{g},\vec{z})\leqslant u(r)\end{cases}.$$

Участок массива A_a, который должен быть принят процессором R от процессора r после выполнения яруса параллельной формы для реализации удаленной записи a в некоторой позиции инструкции S, определяется следующими ограничениями на индексы массива $\vec{g}$:

$$Q_a^{w-r}(\vec{g},R,r):\begin{cases}\vec{g}\in\widetilde{g}_a(\theta',D''_{S,r})\\ l(R)\leqslant\eta_{A_a}(\vec{g},\vec{z})\leqslant u(R)\end{cases}.$$

Определение Многогранниками коммуникаций называются множества $Q_a^{r/w-r/s}$. В первой части верхнего индекса r означает операцию удаленного чтения (read), w — операцию удаленной записи (write). Во второй части верхнего индекса r означает прием данных (receive), s — отправку (send).

Такие многогранники могут быть просканированы гнездами циклов, для генерации которых подходит инструмент cloog [88]. Код этих гнезд можно использовать при формировании пакетов для осуществления информационного обмена между процессами в MPI-коммуникаторе.

Уточним шаг 2 параллельной программы для реализации операций удаленного чтения (листинг 3.1). Отправка данных производится в неблокирующем режиме, чтобы исключить взаимную блокировку процессов. Если размещение вычислений и данных имеет свойство вперед направленных коммуникаций, то возможно использование только функций MPI с блокирующим поведением.

Листинг 3.1 Схема реализации удаленного чтения

```
// Общий случай
for (int j = 1; j <= n; j++) {
  if (a_j is read) {
    for (int r = 0; r < Q; r++)
      MPI_Isend(data for Q_{a_j}^{r-s}(g_{A_{a_j}}, R, r));
    for (int r = 0; r < Q; r++)
      MPI_Recv(data for Q_{a_j}^{r-r}(g_{A_{a_j}}, R, r));
  }
}
```

```
// Свойство FCO выполняется
for (int j = 1; j <= n; j++) {
  if (a_j is read) {
    for (int r = 0; r < R; r++)
      MPI_Recv(data for Q_{a_j}^{r-r}(g_{A_{a_j}}, R, r));
    for (int r = R + 1; r < Q; r++)
      MPI_Send(data for Q_{a_j}^{r-s}(g_{A_{a_j}}, R, r));
  }
}
```

Уточним шаг 5 параллельной программы для реализации операций удаленной записи (листинг 3.2). Отправка данных также производится в

неблокирующем режиме, чтобы исключить взаимную блокировку процессов. Если размещение вычислений и данных имеет свойство вперед направленных коммуникаций, то и здесь возможно использование только функций MPI с блокирующим поведением.

Листинг 3.2 Схема реализации удаленной записи

```
// Общий случай
for (int j = 1; j <= n; j++) {
  if (a_j is write) {
    for (int r = 0; r < Q; r++)
      MPI_Isend(data for Q_{a_j}^{w-s}(g_{A_{a_j}}, R, r));
    for (int r = 0; r < Q; r++)
      MPI_Recv(data for Q_{a_j}^{w-r}(g_{A_{a_j}}, R, r));
  }
}
```

```
// Свойство FCO выполняется
for (int j = 1; j <= n; j++) {
  if (a_j is write) {
    for (int r = 0; r < R; r++)
      MPI_Recv(data for Q_{a_j}^{w-r}(g_{A_{a_j}}, R, r));
    for (int r = R + 1; r < Q; r++)
      MPI_Send(data for Q_{a_j}^{w-s}(g_{A_{a_j}}, R, r));
  }
}
```

Переменная R хранит значение собственного индекса физического процессора (ранг MPI-процесса). Переменная r используется для индексирования других физических процессоров (рангов других MPI-процессов). В приведенных схемах предполагается, что прием или отправка не будут инициированы, если соответствующий многогранник коммуникаций пуст. Это позволяет не выполнять проверку $R \neq r$ для случаев, когда размещение вычислений и данных не обладает свойством вперед направленных коммуникаций.

Рассмотрим вновь реализацию алгоритма LU-разложения квадратной матрицы A (листинг 2.5). Мгновенные домены инструкций и их контексты имеют вид (3.2) при $\theta' = [c_1]$.

$$
C''_{S_1,R}: \begin{cases} N \geqslant 0 \\ c_1 \geqslant 0 \\ c_1 \leqslant 2N-3 \\ c_1 = 2v+1 \\ v \geqslant 0 \end{cases} \qquad C''_{S_2,R}: \begin{cases} N \geqslant 0 \\ c_1 \geqslant 0 \\ c_1 \leqslant 2N-3 \\ c_1 = 2v \\ v \geqslant 0 \end{cases}
$$

$$
D''_{S_1,R}: \begin{cases} l(R) \leqslant c_2 \leqslant u(R) \\ 2c_8 \geqslant c_1 + 1 \\ c_8 \leqslant N-1 \end{cases} \qquad D''_{S_2,R}: \begin{cases} l(R) \leqslant c_2 \leqslant u(R) \end{cases} \tag{3.2}
$$

Доступ r_3 подразумевает удаленное чтение. Процессор, на котором расположена строка матрицы `A[v]`, должен отправить ее фрагмент `A[v][v+1:N-1]` всем

остальным процессорам с большим индексом перед выполнением яруса параллельной формы. Многогранники коммуникаций имеют вид (3.3).

$$\vec{g}_{A_{r_3}} = \begin{bmatrix} (c_1 - 1)/2 \\ c_8 \end{bmatrix} \qquad Q^{r-r}_{r_3}(\vec{g}_{A_{r_3}}, R, r) : \begin{cases} \vec{g}^{(1)}_{A_{r_3}} = v \\ \vec{g}^{(2)}_{A_{r_3}} \geqslant v + 1 \\ \vec{g}^{(2)}_{A_{r_3}} \leqslant N - 1 \\ l(r) \leqslant g^{(1)}_{A_{r_3}} \leqslant u(r) \end{cases}$$

$$\widetilde{g}_{r_3}(\theta', D''_{S_1,R}) : \begin{cases} \vec{g}^{(1)}_{A_{r_3}} = v \\ \vec{g}^{(2)}_{A_{r_3}} \geqslant v + 1 \\ \vec{g}^{(2)}_{A_{r_3}} \leqslant N - 1 \end{cases} \qquad Q^{r-s}_{r_3}(\vec{g}_{A_{r_3}}, R, r) : \begin{cases} \vec{g}^{(1)}_{A_{r_3}} = v \\ \vec{g}^{(2)}_{A_{r_3}} \geqslant v + 1 \\ \vec{g}^{(2)}_{A_{r_3}} \leqslant N - 1 \\ l(R) \leqslant g^{(1)}_{A_{r_3}} \leqslant u(R) \end{cases} \tag{3.3}$$

Доступ r_5 также подразумевает удаленное чтение. Процессор, на котором расположен элемент матрицы `A[v][v]`, должен отправить его всем остальным процессорам с большим индексом перед выполнением яруса параллельной формы. Многогранники коммуникаций имеют вид (3.4).

$$\vec{g}_{A_{r_5}} = \begin{bmatrix} c_1/2 \\ c_1/2 \end{bmatrix} \qquad Q^{r-r}_{r_5}(\vec{g}_{A_{r_5}}, R, r) : \begin{cases} \vec{g}^{(1)}_{A_{r_5}} = v \\ \vec{g}^{(2)}_{A_{r_5}} = v \\ l(r) \leqslant g^{(1)}_{A_{r_5}} \leqslant u(r) \end{cases}$$

$$\widetilde{g}_{r_5}(\theta', D''_{S_2,R}) : \begin{cases} \vec{g}^{(1)}_{A_{r_5}} = v \\ \vec{g}^{(2)}_{A_{r_5}} = v \end{cases} \qquad Q^{r-s}_{r_5}(\vec{g}_{A_{r_5}}, R, r) : \begin{cases} \vec{g}^{(1)}_{A_{r_5}} = v \\ \vec{g}^{(2)}_{A_{r_5}} = v \\ l(R) \leqslant g^{(1)}_{A_{r_5}} \leqslant u(R) \end{cases} \tag{3.4}$$

Остальные доступы к памяти не подразумевают операций удаленного чтения или записи, поэтому для них не требуется строить многогранники коммуникаций. Доступ к строкам матрицы всегда локальный, о чем свидетельствует первое неравенство в определениях (3.5). Многогранники коммуникаций для доступов r_3 и r_5 представляют фрагмент строки матрицы. Если расположение матрицы в памяти выполнено согласно линеаризации по строкам, то информационный пакет для приема или передачи может обрабатываться тривиально как непрерывный участок памяти непосредственными вызовами функций `MPI_Send` и `MPI_Recv` (или их неблокирующими аналогами).

$$
\begin{aligned}
\vec{g}_{A_{r_1,w_1}} &= \begin{bmatrix} c_2 \\ c_8 \end{bmatrix} & \widetilde{g}_{r_1,w_1}(\theta', D''_{S_1,R}) &: \begin{cases} l(R) \leqslant \vec{g}^{(1)}_{A_{r_1,w_1}} \leqslant u(R) \\ 2g^{(2)}_{A_{r_1,w_1}} \geqslant c_1 + 1 \\ g^{(2)}_{A_{r_1,w_1}} \leqslant N - 1 \end{cases} \\
\vec{g}_{A_{r_2}} &= \begin{bmatrix} c_2 \\ (c_1 - 1)/2 \end{bmatrix} & \widetilde{g}_{r_2}(\theta', D''_{S_1,R}) &: \begin{cases} l(R) \leqslant \vec{g}^{(1)}_{A_{r_2}} \leqslant u(R) \\ \vec{g}^{(2)}_{A_{r_2}} = v \end{cases} \\
\vec{g}_{A_{r_4}} &= \begin{bmatrix} c_2 \\ c_1/2 \end{bmatrix} & \widetilde{g}_{r_4}(\theta', D''_{S_2,R}) &: \begin{cases} l(R) \leqslant g^{(1)}_{A_{r_4}} \leqslant u(R) \\ \vec{g}^{(2)}_{A_{r_4}} = v \end{cases}
\end{aligned}
\tag{3.5}
$$

В общем случае многогранники коммуникаций, являясь выпуклыми множествами, могут определять информационные пакеты, составленные несколькими непрерывными областями памяти. Очевидный пример — столбец матрицы, линеаризованной по строкам. Разработана библиотека макросов blockdist.h, предоставляющая средства для работы с многогранниками коммуникаций в форме фрагментов строк и столбцов матрицы, линеаризованной по строкам, и тем самым упрощающая обработку таких нерегулярных структур, как области матриц произвольной формы, также разработаны процедуры для локализации результатов вычислений в главном процессе приложения (приложение А.1).

3.2 Постобработка программного кода на языке C

В cloog [88] версии 0.18.4 внесены изменения для специальной обработки цикла, если его индекс имеет имя `ilpp`. Этот индекс далее используется как итератор по пространству виртуальных процессоров. Специальная обработка заключается в том, чтобы этот цикл всегда отображался в результирующей параллельной программе, даже если его конструкция подразумевает всего одну итерацию. Это упрощает автоматизацию аннотирования параллельных циклов директивами OpenMP, а также их подготовку для MPI в рамках блочной схемы распределения процессоров. Часть изменений программного кода на языке C сводится к тривиальным текстовым вставкам и заменам.

3.2.1 Расстановка директив OpenMP

Весь код, сгенерированный cloog, помещается в параллельный регион OpenMP с помощью директивы `#pragma omp parallel`. Параллельные циклы, имеющие только одну итерацию, помечаются директивой `#pragma omp master`, направляющей вычисления в главную нить без барьерной синхронизации. Параллельные циклы, имеющие более одной итерации, помечаются сразу двумя директивами: во-первых, устанавливается явная барьерная синхронизация `#pragma omp barrier`, чтобы синхронизировать главную нить с остальными перед выполнением параллельного цикла, во-вторых, отмечается распараллеливание цикла по нитям внутри параллельного региона с помощью `#pragma omp for`. Неявная барьерная синхронизация устанавливается после выполнения параллельного цикла. Таким образом, синхронизационные барьеры будут обрамлять только параллельные циклы с количеством итераций больше одной. Скрипт omp_annotate.py (листинг 3.3) выполняет расстановку директив OpenMP, получая неразмеченную программу на стандартный ввод, и выдавая результат на стандартный вывод. В заголовки циклов добавляется тип счетчика int для соответствия стандарту C99.

Листинг 3.3 Скрипт аннотирования директивами OpenMP omp_annotate.py

```
#!/usr/bin/python3
import sys
import re
indentation = "    "
def decorated_print(s):
    print(indentation + s.replace("  ", "    "), end="")
decorated_print("#pragma omp parallel\n")
decorated_print("{\n")
for line in sys.stdin:
    line_strip = line.strip()
    if re.match("for \(ilpp=", line_strip):
        l = line_strip.split()
        lh = l[1].split(";")
        start = lh[0].split("=")[1]
        fin = lh[1].split("<=")[1]
        spaces = " " * line.index("for")
        if start == fin:
            decorated_print(spaces + "  #pragma omp master\n")
        else:
            decorated_print(spaces + "  #pragma omp barrier\n")
            decorated_print(spaces + "  #pragma omp for\n")
    decorated_print("  " + line.replace("for (", "for (int "))
decorated_print("}\n")
```

3.2.2 Подготовка параллельных циклов для MPI

Подготовка параллельных циклов для MPI подразумевает изменение границ изменения счетчика `ilpp`, чтобы для каждого физического процессора он не выходил за границы сопоставленного отрезка пространства виртуальных процессоров (от `lR` до `uR` включительно) согласно схеме блочного распределения процессоров. Начальное значение `start` заменяется на `max(lR,start)`, конечное значение `fin` заменяется на `min(uR,fin)`. Также после каждого параллельного цикла вставляется явная барьерная синхронизация `MPI_Barrier(MPI_COMM_WORLD)`. Скрипт mpi_annotate.py (листинг 3.4) выполняет обработку параллельных циклов, получая исходную программу на стандартный ввод, и выдавая результат на стандартный вывод. В заголовки циклов добавляется тип счетчика int для соответствия стандарту C99.

Листинг 3.4 Подготовка параллельных циклов для MPI mpi_annotate.py

```
#!/usr/bin/python3
import sys
import re
indentation = "    "
def decorated_print(s):
    print(indentation + s.replace("  ", "    "), end="")
rank = 0
synced = True
for line in sys.stdin:
    line_strip = line.strip()
    if re.match("for \(ilpp=", line_strip):
        l = line_strip.split()
        lh = l[1].split(";")
        start = lh[0].split("=")[1]
        fin = lh[1].split("<=")[1]
        spaces = " " * line.index("for")
        line = spaces + "for (ilpp=max(lR," + start + ");ilpp<=min(uR," + fin + ");" + lh[2] + " {\n"
        synced = False
    decorated_print(line.replace("for (", "for (int "))
    if (not synced):
        rank = rank + line.count("{") - line.count("}")
        if (rank == 0):
            synced = True
            decorated_print(spaces + "MPI_Barrier(MPI_COMM_WORLD);\n")
```

Как только в тексте встречается параллельный цикл, флаг `synced` сбрасывается, что означает необходимость найти конец параллельного цикла и вставить барьерную синхронизацию. Конец параллельного цикла находится путем отслеживания ранга заведомо корректной скобочной последовательности, состоящей

из открывающих и закрывающих фигурных скобок, в переменной `rank`. Как только ранг принимает нулевое значение, встречен конец параллельного цикла: добавляется барьерная синхронизация и флаг `synced` взводится, что означает завершение анализа скобочной последовательности.

3.2.3 Расстановка конструкций информационного обмена

Приводится методика расстановки конструкций информационного обмена, реализованных в библиотеке blockdist.h.

Реализация удаленного чтения вставляется перед параллельным циклом, а реализация удаленной записи — после него. Каждый информационный обмен обрамляется комментариями:

```
// доступ in инструкция
if ограничения контекста {
  макросы blockdist.h
}
//
```

Ограничения контекста выписываются как конъюнкция условий из определения контекста мгновенного домена инструкции. Не все из конъюнктов являются необходимыми: ограничения, соответствующие границам изменения счетчиков циклов выше по тексту, стоит пропустить, так как само наличие цикла накладывает их автоматически. Также могут быть пропущены ограничения на внешние переменные программы, так как их проверка имеет смысл до начала всех вычислений. Оператор ветвления вписывается только в том случае, если остался хотя бы один конъюнкт. Для двух удаленных доступов, не породивших ни одного конъюнкта, может быть выполнена оптимизация: если многогранник коммуникаций одного доступа полностью включает многогранник коммуникаций другого, то требуется только один акт информационного обмена. В этом случае в текст программы добавляется только строка комментария перед макросами blockdist.h, несущая информацию о доступе, для которого не потребовался отдельный акт информационного обмена. Если параллельный цикл является телом цикла в плотно вложенном гнезде циклов выше по тексту, то допускается исключить индексы циклов гнезда из мгновенного домена рассматриваемой инструкции с доступом к памяти, и выполнять информационный обмен в окрестности всего гнезда циклов, если это приведет к складыванию многогранников коммуникаций в непрерывные

структуры, и тем самым сократит количество актов информационного обмена. Последний параллельный цикл на листингах В.16 и В.17 позволяет выполнить эту оптимизацию.

Тело параллельного цикла просматривается сверху вниз по тексту, тело инструкций просматривается слева направо. Это определяет последовательность вставки конструкций информационного обмена для каждого удаленного доступа перед параллельным циклом или после него.

Для каждого доступа к массиву выполняется следующий статический анализ: проверяется, совпадает ли индекс виртуального процессора, владеющего элементом массива, с индексом виртуального процессора, исполняющего динамический экземпляр инструкции, в которой встретился рассматриваемый доступ. На практике проверяется, является ли арифметическое выражение, определяющее размещение элемента массива в согласии с найденным размещением данных, тождественно равным `ilpp`. Если является, то доступ рассматривается как локальный, в противном случае — как удаленный. В главе 4 будут рассмотрены примеры, в которых характер доступа будет ясен по результатам применения разработанного метода нахождения пространственных и временных отображений программ: $L_a^{\rho} = 0$ сигнализирует о том, что доступ a — локальный.

Приоритет выбора макросов для реализации операции удаленного доступа следующий:

1. Если выполняется свойство вперед направленных коммуникаций:
   а) `[line|col]_Q_[read|write]_vp[_rev]_fco`.
   б) `[line|col]_Q_[read|write]_[send|receive]_fco`.
2. Макросы для общего случая:
   а) `[line|col]_Q_[read|write]_vp[_rev]`.
   б) `[line|col]_Q_[read|write]_[send|receive]`.

Если применимы макросы с суффиксом `_vp`, их использование предпочтительнее, поскольку они уже включают барьерную синхронизацию наряду с отправкой и приемом данных. При использовании отдельных универсальных конструкций для приема и отправки данных барьерная синхронизация должна быть вписана явно.

При обработке строк матриц с применением разработанных макросов с префиксом `line_` требуется для каждого удаленного доступа задать макрос с одним параметром, определяющий размещение заданного элемента фиксированной строки матрицы, к которой выполняется доступ. Это необходимо для обеспече-

ния совместимости с параметром `eta`, который трактуется как макрос с одним параметром, поскольку макросы с префиксом `line_` ориентированы в первую очередь на обработку линейных массивов. Если имеется макрос размещения матрицы `eta_A`, то новый макрос размещения элемента строки `eta_A_access` задается с его помощью:

```
1 #define eta_A(i0,i1) (data_placement)
2 #define eta_A_access(i1) eta_A((row_number),i1)
```

`data_placement` является аффинным выражением от `i0` (строка матрицы), `i1` (столбец матрицы) и внешних переменных программы, представляя найденное размещение данных. `row_number` — арифметическое выражение номера строки матрицы, скопированное из тела доступа к элементу массива.

При передаче строки или столбца матрицы рекомендуется сообщения помечать различными тэгами, задавая параметр `tag` как $T_A + i$, где T_A — базовый тэг массива `A`, i — индекс передаваемой строки или столбца матрицы. Базовый тэг массива `A` можно определить следующим образом: $T_A = I_A \cdot T_{base}$, где I_A — порядковый номер массива начиная с единицы (может быть задан произвольно или определен при обработке представления программы в OpenSCOP [111]), T_{base} — достаточно большое положительное число, чтобы выполнялось $T_A + i < T_A + T_{base}$ для каждого массива `A`, что позволит избежать совпадений тэгов для различных массивов.

3.2.4 Оптимизация ветвлений

Библиотека cloog генерирует циклы следующего вида:

```
1  for (ilpp=start;ilpp<=fin;ilpp++) {
2    if (c_1) {
3      code_1;
4    }
5    if (c_2) {
6      code_2;
7    }
8    ...
9    if (c_N) {
10     code_N;
11   }
12 }
```

Рассмотрим частный случай, когда условия c_i, $i = 1, \ldots, N$ накладывают ограничения только на индексы циклов выше по тексту (в соответствии с

мгновенными доменами каждой из инструкций, встречающейся в телах операторов ветвления $code_i$, $i = 1, \ldots, N$), при этом некоторые из этих условий могут быть идентичны, но никакие два различных не могут выполняться одновременно. Удалим повторяющиеся условия, изменив поток управления в рассматриваемом фрагменте программы. Обозначим теперь уже уникальные условия g_j, $j = 1, \ldots, M$. Каждая группа $code_{g_j}$, $j = 1, \ldots, M$ состоит из фрагментов $code_i$, $i \in \{i\}_{g_j}$, обрамленных условием g_j. Порядок фрагментов в группе сохраняется относительно исходного текста: если, например, $c_2 = c_4 = g_2$, то $code_{g_2}$ будет составлен из $code_2$ и $code_4$ именно в порядке возрастания индексов.

```
for (ilpp=start;ilpp<=fin;ilpp++) {
  if (g_1) {
    code_{g_1};
  }
  if (g_2) {
    code_{g_2};
  }
  ...
  if (g_M) {
    code_{g_M};
  }
}
```

Чтобы исключить обработку одних и тех же условий на каждой итерации параллельного цикла, можно вынести операторы ветвления за заголовок цикла:

```
if (g_1) {
  for (ilpp=start;ilpp<=fin;ilpp++) {
    code_{g_1};
  }
}
if (g_2) {
  for (ilpp=start;ilpp<=fin;ilpp++) {
    code_{g_2};
  }
}
...
if (g_M) {
  for (ilpp=start;ilpp<=fin;ilpp++) {
    code_{g_M};
  }
}
```

Это преобразование имеет смысл, если параллельный цикл содержит более одной итерации — только в этом случае условия будут обрабатываться меньшее количество раз (единожды перед выполнением параллельного цикла).

На рисунке 3.1 приведен пример результата компиляции трех ветвлений в среде Compiler Explorer [112]. Версия компилятора gcc и ключи оптимизации соответствуют тестовой среде (глава 4). Первая проверка четности реализуется двумя инструкциями процессора, вторая — тремя, а третья проверка кратности

— девятью. Именно на обработку проверок, требующих для реализации большего числа инструкций процессора, чем проверка четности, и нацелена оптимизация ветвлений в постобработке кода.

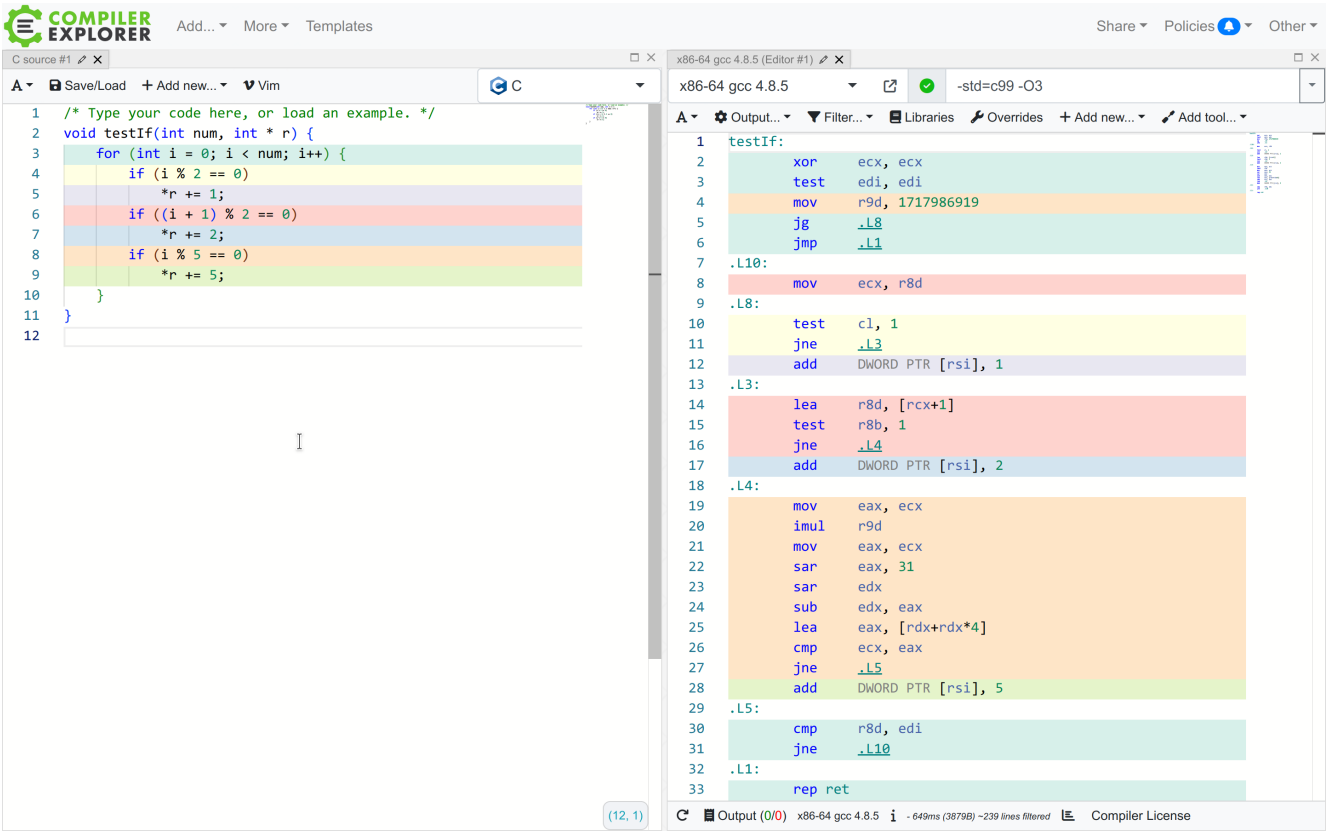


Рисунок 3.1 — Обработка ветвлений компилятором gcc

3.3 Транслятор ilpy: архитектура и возможности

Разработанный метод нахождения пространственных и временных отображений программ линейного класса (глава 2) был доведен до практического воплощения в программной реализации [43] в виде транслятора текст-в-текст. Созданное программное обеспечение транслирует исходную последовательную программу на языке C в ее оптимизированный для параллельного выполнения вариант также на языке C. Дальнейшие отсылки к разработанному транслятору будут включать его название ilpy (от аббревиатуры ILP — Integer Linear Programming).

Программная реализация ilpy выполнена на языке C в соответствии с принципами структурного программирования и включает в себя следующие модули:

1. `dependence_polyhedron` — структура данных для представления многогранника зависимостей (с разбиением списка ограничений на секции).
2. `affine_mapping` — структура данных для работы с аффинными отображениями, в частности для их пересчета из найденных множителей Фаркаша.
3. `farkas` — структуры данных для представления инструкций, массивов, зависимостей по данным, доступов к данным, а также подпрограммы для применения леммы Фаркаша и составления систем ограничений для задач ЛЦП.
4. `glp_utils` — функции для манипулирования строками матрицы, представляющей ограничения в задаче ЛЦП.
5. `matrix_operations` — операции с матрицами вещественных чисел двойной точности, применяемые для нахождения размещений вычислений (линейно независящих от компонентов многомерного расписания).
6. `polyhedral_operations` — операции с многогранниками: вычисление количества точек с целочисленными координатами, вычисление образа многогранника по аффинной функции.
7. `statement_scattering` — структура данных для представления всего набора аффинных отображений, сопоставленных инструкции.
8. `schedule` — нахождение расписаний вычислений.
9. `placement` — нахождение размещений вычислений и данных.
10. `reporting` — вывод найденных аффинных отображений (расписаний θ, размещений вычислений π и данных η) и диагностической информации (аффинных функций L_e^τ, L_e^ρ, L_a^ρ).
11. `main` — реализация потока данных для выполнения трансляции.

Все этапы распараллеливания программы с применением ilpy включены в диаграмму потоков данных на рисунке 3.2.

Получение параллельного кода без аннотаций OpenMP и конструкций MPI реализуется транслятором ilpy с применением сторонних библиотек и их модификаций:

– osl [111] используется для представления полиэдрального представления программы в памяти и взаимодействия с библиотеками clan, candl, cloog;

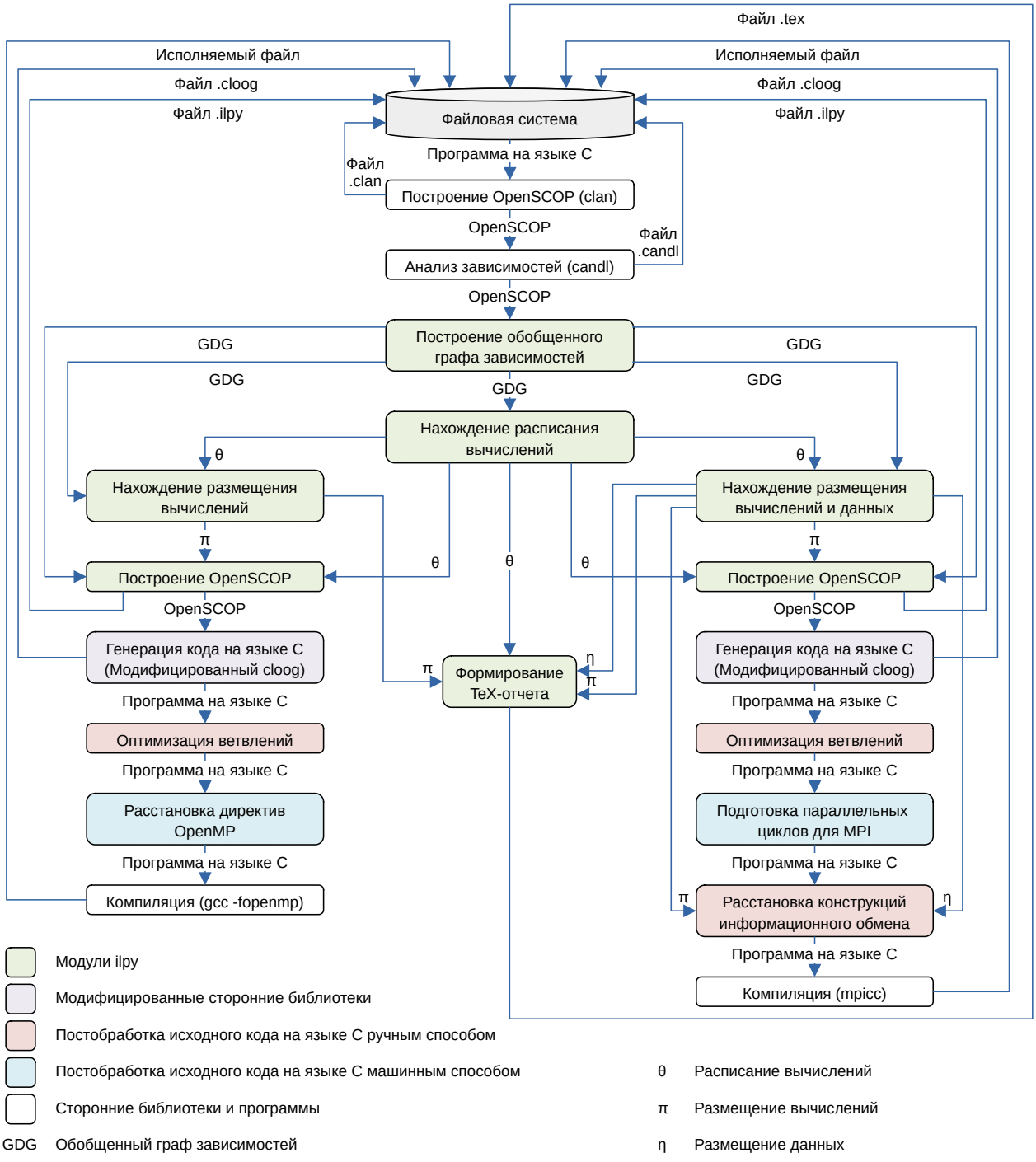


Рисунок 3.2 — Трансляция линейной программы с применением ilpy

– clan [113] используется для извлечения полиэдрального представления программы OpenSCOP;
– candl [36] используется для расширения полиэдрального представления программы OpenSCOP информацией о многогранниках зависимостей;

– cloog [88] используется в модифицированном виде для генерации параллельных программ на языке C в соответствии с найденными аффинными отображениями;
– polylib [108] используется для вычисления количества точек с целочисленными координатами внутри многогранника (алгоритм вычисляет значение полиномов Эрхарта [114]), а также для вычисления образа многогранника по аффинной функции;
– glpk [115] используется для решения задач линейного целочисленного программирования;
– openblas [116] используется для реализации матричной арифметики.

Отношение зависимости модулей иллюстрирует диаграмма на рисунке 3.3.

Постобработка полученного параллельного кода включает преобразования, выполняемые машинным способом (расстановка директив OpenMP и подготовка параллельных циклов для MPI), а также преобразования, выполняемые ручным способом: оптимизация ветвлений и расстановка конструкций информационного обмена blockdist.h. Финальная компиляция выполняется с помощью компилятора gcc или обертывающей команды mpicc.

Транслятор ilpy предоставляет интерфейс командной строки:

```
$ ilpy
Usage: ilpy-core <filename.c> [--out-fn-suffix <string>] [--arrays] [--async] [--smdp]
[--clooog-f <int>] [--cloog-l <int>] [--ilp-col-ub <double>] [--iocp-tm-lim <int>]
[--param-weight <param_name>=<int>[(,<param_name>=<int>)+]]
```

Первый и обязательный параметр — имя файла, содержащего линейную программу. Директивы препроцессора `#pragma scop` и `#pragma endscop` отмечают начало и конец фрагмента, удовлетворяющего свойствам линейной программы. Эти директивы игнорируются компиляторами в нормальном режиме работы. В процессе работы транслятора будут созданы следующие файлы:

– `<filename.c>.clan` — представление программы в формате OpenSCOP, полученное применением clan.
– `<filename.c>.candl` — представление программы в формате OpenSCOP, расширенное зависимостями по данным, полученное применением candl.
– `<filename.c>-<string>.ilpy` — представление программы в формате OpenSCOP с аффинными отображениями, найденными ilpy.
– `<filename.c>-<string>.cloog` — параллельная программа на языке C, сгенерированная cloog на основе найденных ilpy аффинных отображений.

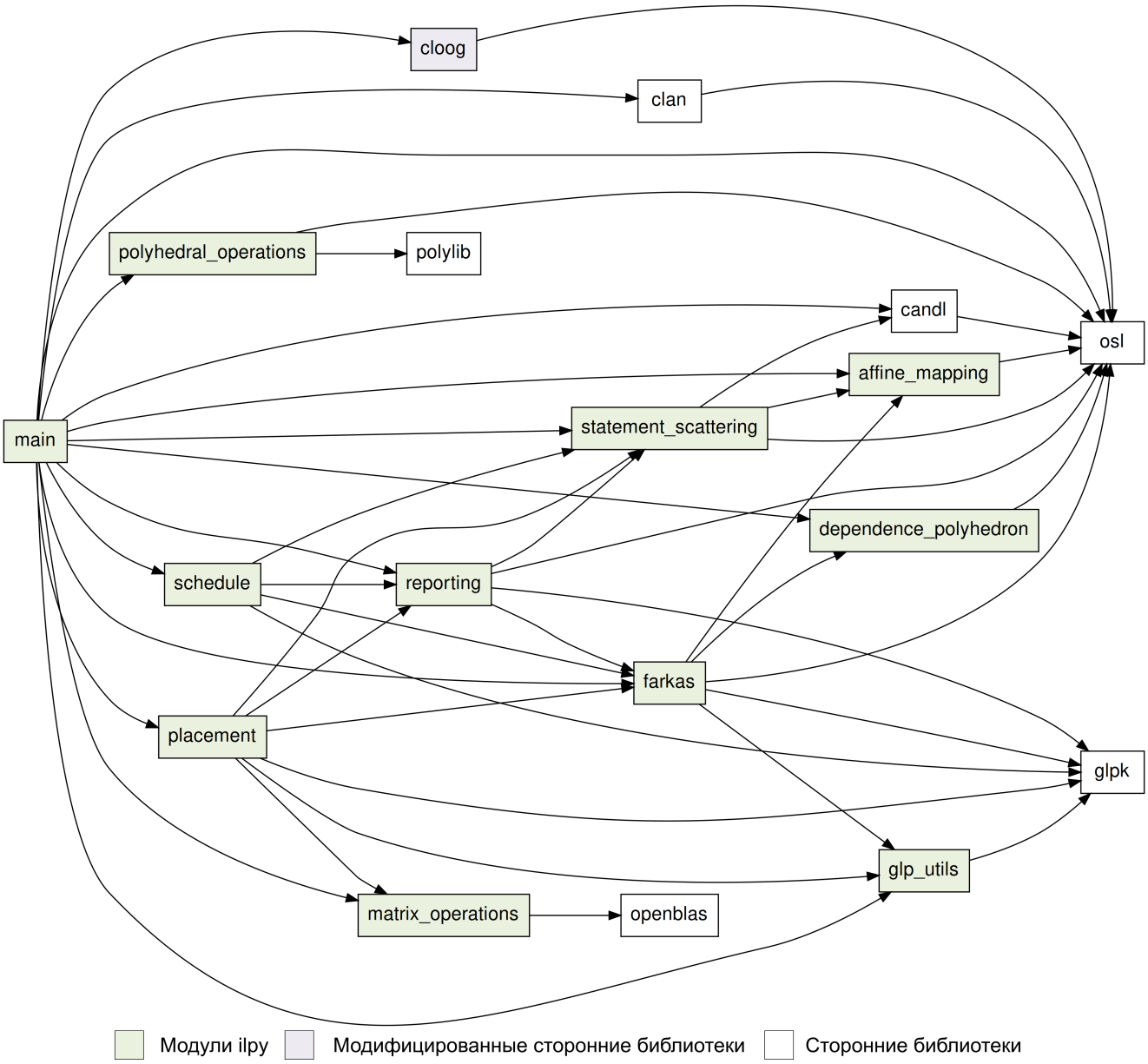


Рисунок 3.3 — Диаграмма зависимостей модулей ilpy

– `<filename.c>-<string>.tex` — отчет о найденных аффинных отображениях в исходном тексте LaTeX, содержащем формулы аффинных отображений.

Суффикс имени выходных файлов `.ilpy`, `.cloog` и `.tex` задается необязательным параметром `--out-fn-suffix <string>`. Если параметр не задан, используется значение по умолчанию: `arrays`, если задан параметр `--arrays`, иначе `adeps`, если задан параметр `--async`, иначе `deps`.

Следующие параметры также являются необязательными:

– `--arrays` — выполнить распараллеливание для MPI (вычисляется размещение данных). Если параметр не задан, то по умолчанию предполагается

распараллеливание для OpenMP и вычисление размещения данных не производится.

– `--async` — сконструировать (возможно, некорректную) программу с асинхронным параллелизмом. При этом вычисляется многомерное размещение вычислений (и данных, если задан параметр `--arrays`) по аналогии с тем, как выполняется построение линейно независимых аффинных отображений в pluto. Рассмотренный в главе 4 пример программы syr2k показывает, что такой подход может повысить производительность итоговой параллельной программы, если она содержит гнездо с переставляемыми циклами. Если параметр не задан, то вычисляется одномерное размещение вычислений и данных и формируется корректная программа с синхронным параллелизмом.
– `--smdp` — применить топологическую сортировку для вычисления первого компонента многомерного расписания, если обобщенный граф зависимостей не содержит циклов. Рассмотренный в главе 4 пример программы atax показывает, что такой подход может дать выгодные с точки зрения производительности итоговой параллельной программы аффинные отображения. Если параметр не задан, то по умолчанию все компоненты многомерного расписания вычисляются как оптимальные наборы аффинных отображений.
– `--cloog-f <int>` — параметр cloog «First Depth to Optimize Control» — начальная глубина цикла, на которой начинается оптимизация потока управления. Чем меньше это значение, тем меньше накладных расходов на организацию потока управления, но длиннее генерируемый код. Минимальное значение — 1, максимальное значение — бесконечность (кодируется как -1). Значение по умолчанию в cloog — 1. Значение по умолчанию в ilpy — -1, то есть по умолчанию оптимизации cloog в ilpy отключены.
– `--cloog-l <int>` — параметр cloog «Last Depth to Optimize Control» — конечная глубина цикла, на которой завершается оптимизация потока управления. Чем больше это значение, тем меньше накладных расходов на организацию потока управления. Максимальное значение — бесконечность (кодируется как -1). Значение по умолчанию в cloog и ilpy — -1.
– `--ilp-col-ub <double>` — ограничить множители Фаркаша и переменные-ограничители в формулировке задачи ЛЦП для вычисле-

ния размещений вычислений (и данных, если задан параметр `--arrays`) сверху заданным положительным вещественным числом. Это может сократить время работы решателя, так как сужает область поиска.

– `--iocp-tm-lim <int>` — ограничить время работы решателя задачи ЛЦП для вычисления размещений вычислений (и данных, если задан параметр `--arrays`) заданным количеством секунд.
– `--param-weight <param_name>=<int>` — задать вес параметра с именем `param_name` целым неотрицательным числом. Разрешается повторение конструкций через запятую. По умолчанию каждому параметру присваивается вес 100. Можно передать информацию о контексте (соотношении параметров программы), задавая различные веса. Рассмотренный в главе 4 пример программы gramschmidt иллюстрирует необходимость введения этой опции.

Для рассмотренных в главе 4 программ lu, syr2k, floyd оптимизации cloog не были включены за отсутствием необходимости: параллельные варианты floyd и syr2k вовсе не имеют ветвлений, а параллельный вариант lu содержит тесты на четность индекса цикла, которые эффективно обрабатываются современными компиляторами, в отличие от более сложных проверок с делением, которые могут появиться в коде даже при включенных оптимизациях cloog.

Параметры `--ilp-col-ub` и `--iocp-tm-lim` нацелены на снижение времени работы решателя glpk при решении задачи ЛЦП для вычисления размещений вычислений (и данных, если задан параметр `--arrays`) за счет отказа от поиска оптимума в том смысле, который отражен в (2.5) и (2.15).

3.3.1 Полиэдральное представление программы OpenSCOP

OpenSCOP [111] — это открытая спецификация, которая определяет формат файла и набор структур данных для представления кода со статическим потом управления в модели многогранников. Цель разработки OpenSCOP — предоставить общий интерфейс для различных инструментов преобразования программ в модели многогранников, чтобы упростить их взаимодействие. Применяемые в работе инструменты clan, candl, cloog используют библиотеку osl, реализующую спецификацию OpenSCOP, для взаимодействия с внешними программами и фай-

лового ввода-вывода. ilpy так же использует osl для взаимодействия с clan, candl, cloog и файлового вывода (файлы .clan, .candl, .ilpy).

Ключевой абстракцией OpenSCOP является отношение (relation), с помощью которой может быть представлен многогранник. Многогранник может описывать домен инструкции, ограничения на внешние переменные программы (контекст), доступ к массиву на чтение и запись, зависимость по данным, аффинное отображение. Представление многогранника включает:

1. Тип отношения:
   - `UNDEFINED` — отношение общего вида,
   - `CONTEXT` — ограничения контекста,
   - `DOMAIN` — домен инструкции,
   - `SCATTERING` — аффинные отображения,
   - `READ` — доступ к массиву на чтение,
   - `WRITE` — доступ к массиву на запись.
2. Описание матрицы ограничений:
   а) количество строк,
   б) количество столбцов,
   в) количество выходных измерений,
   г) количество входных измерений,
   д) количество локальных измерений,
   е) количество параметров.

   Сумма последних четырех чисел должна быть равна количеству столбцов минус два. Оставшиеся два столбца — индикатор равенства/неравенства и постоянный член. Количество параметров должно быть одинаковым для всех отношений во всем файле или структуре данных OpenSCOP.
3. Строки матрицы ограничений. Ограничение представляет равенство $p(x) = 0$, если первый элемент строки содержит ноль, или неравенство $p(x) \geqslant 0$, если первый элемент строки содержит 1. Следующие элементы являются коэффициентами выходных измерений, за которыми следуют коэффициенты входных измерений, затем локальных измерений, и, наконец, параметров. Последний элемент является постоянным членом.

Приведем примеры отношений для программы LU-разложения квадратной матрицы A из пакета Polybench [110], приведенной на листинге 2.1. Фрагменты текста, начинающиеся с символа #, являются комментариями.

Домен D_{S_1} в последовательной программе имеет описание:

```
DOMAIN
6 5 2 0 0 1
# e/i|  k    l |  N |  1
   1    1    0    0    0    ## k >= 0
   1   -1    0    1   -1    ## -k+N-1 >= 0
   1    0    0    1   -1    ## N-1 >= 0
   1   -1    1    0   -1    ## -k+l-1 >= 0
   1    0   -1    1   -1    ## -l+N-1 >= 0
   1   -1    0    1   -2    ## -k+N-2 >= 0
```

Аффинные отображения для S_1 соответствуют потоку управления:

```
SCATTERING
5 10 5 2 0 1
# e/i| c1   c2   c3   c4   c5 |  k    l |  N |  1
   0   -1    0    0    0    0    0    0    0    0    ## c1 == 0
   0    0   -1    0    0    0    1    0    0    0    ## c2 == k
   0    0    0   -1    0    0    0    0    0    0    ## c3 == 0
   0    0    0    0   -1    0    0    1    0    0    ## c4 == l
   0    0    0    0    0   -1    0    0    0    0    ## c5 == 0
```

Доступы к массивам в инструкции S_1 также описываются в матричном виде:

```
READ
3 8 3 2 0 1
# e/i| Arr  [1]  [2]|  k    l |  N |  1
   0   -1    0    0    0    0    0    4    ## Arr == A
   0    0   -1    0    0    1    0    0    ## [1] == l
   0    0    0   -1    1    0    0    0    ## [2] == k
WRITE
3 8 3 2 0 1
# e/i| Arr  [1]  [2]|  k    l |  N |  1
   0   -1    0    0    0    0    0    4    ## Arr == A
   0    0   -1    0    0    1    0    0    ## [1] == l
   0    0    0   -1    1    0    0    0    ## [2] == k
READ
3 8 3 2 0 1
# e/i| Arr  [1]  [2]|  k    l |  N |  1
   0   -1    0    0    0    0    0    4    ## Arr == A
   0    0   -1    0    1    0    0    0    ## [1] == k
   0    0    0   -1    1    0    0    0    ## [2] == k
```

Каждому массиву и каждой переменной присваивается целочисленный идентификатор, который и фигурирует в первой строке матрицы ограничений в последнем столбце. Информация об идентификаторах содержится в отдельном расширении OpenSCOP arrays:

```
<arrays>
# Number of arrays
6
# Mapping array-identifiers/array-names
1 k
2 N
3 l
4 A
5 i
6 j
</arrays>
```

Пример матрицы ограничений для представления домена инструкции S_1 в параллельном варианте LU-разложения квадратной матрицы A (листинг 2.5) использует локальное измерение для задания только нечетных индексов c1:

```
7 7 3 0 1 1 # 7 строк, 7 столбцов: 3 выходных, 0 входных, 1 локальное, 1 параметр
# e/i| c1   c2   c8 | ld |  N |  1
   1    1    0    0    0    0    0 ## c1 >= 0
   1   -1    0    0    0    2   -3 ## c1 <= 2N-3
   0    1    0    0    2    0    1 ## c1 = 2*ld + 1
   1   -1    2    0    0    0   -1 ## 2*c2 >= c1 + 1
   1    0   -1    0    0    1   -1 ## c2 <= N-1
   1   -1    0    2    0    0   -1 ## 2*c8 >= c1 + 1
   1    0    0   -1    0    1   -1 ## c8 <= N-1
```

Сходным образом задаются и аффинные отображения для этой же инструкции S_1, найденные при распараллеливании:

```
SCATTERING
2 7 2 2 0 1
# e/i| c1   c2 |  k    l |  N |  1
   0   -1    0    2    0    0    0    ## c1 == 2*k
   0    0   -1    0    1    0    0    ## c2 == l
```

3.3.2 Анализ зависимостей по данным и обработка многогранников зависимостей

Расширение dependence спецификации OpenSCOP позволяет представлять многогранники зависимостей отношениями общего вида. В таблице 1 представлена структура матрицы ограничений такого отношения для некоторого ребра e в обобщенном графе зависимостей. Строки матрицы ограничений сгруппированы в шесть секций (перечисление сверху вниз):

1. домен начальной вершины $\sigma(e)$, описываемый ограничениями на целочисленный вектор индекса итерации $\vec{i}_{\sigma(e)}$, могут использоваться локальные измерения;
2. домен конечной вершины $\delta(e)$, описываемый ограничениями на целочисленный вектор индекса итерации $\vec{i}_{\delta(e)}$, могут использоваться локальные измерения;
3. индексная функция доступа в инструкции $\sigma(e)$, зависящая от $\vec{i}_{\sigma(e)}$ и внешних переменных программы $\vec{z}$, описывается уравнениями с возможным привлечением локальных измерений;

4. индексная функция доступа в инструкции $\delta(e)$, зависящая от $\vec{i}_{\delta(e)}$ и внешних переменных программы $\vec{z}$, описывается уравнениями с возможным привлечением локальных измерений;
5. $1 + p_A$ уравнений, фиксирующих совпадение адресов ячейки памяти, к которой осуществляется доступ в инструкциях $\sigma(e)$ и $\delta(e)$: одно уравнение для номера массива A, остальные p_A — для его индексов;
6. p_e уравнений и одно неравенство для описания лексикографического предшествования зависимых операций.

Участие переменных в уравнениях и неравенствах отмечено знаком «+» в ячейках таблицы, сигнализирующим о возможном присутствии соответствующей переменной с ненулевым коэффициентом в уравнении или неравенстве. Заранее известные коэффициенты вписаны явно в виде скаляров и подматриц, а пустые ячейки таблицы соответствуют заполнению участков матрицы ограничений нулями.

Таблица 1 — Представление многогранника зависимостей в OpenSCOP

			Измерения									
			Выходные		Входные		Локальные					
		$1\colon p(x) \geqslant 0$ $0\colon p(x) = 0$	$\vec{i}_{\sigma(e)}$	$\vec{g}_{a_{\sigma(e)}}$	$\vec{i}_{\delta(e)}$	$\vec{g}_{a_{\delta(e)}}$	$D_{\sigma(e)}$	$g_{a_{\sigma(e)}}$	$D_{\delta(e)}$	$g_{a_{\delta(e)}}$	$\vec{z}$	1
$D_{\sigma(e)}$		1	+				+				+	+
$D_{\delta(e)}$		1			+				+		+	+
$g_{a_{\sigma(e)}}(\vec{i}_{\sigma(e)}, \vec{z})$		0	+	+				+			+	+
$g_{a_{\delta(e)}}(\vec{i}_{\delta(e)}, \vec{z})$		0			+	+				+	+	+
$\vec{g}_{a_{\sigma(e)}} = \vec{g}_{a_{\delta(e)}}$		0		I_{1+p_A}		$-I_{1+p_A}$						
$\vec{i}_{\sigma(e)} \underset{lex}{<} \vec{i}_{\delta(e)}$	$1, \ldots, p_e$	0	I_{1+p_e}		$-I_{1+p_e}$							0
	$p_e + 1$	1										0 | -1

Описание зависимости по данным в OpenSCOP содержит следующие секции:

1. Тип: RAW (чтение после записи), WAR (запись после чтения), WAW (запись после записи).
2. Начальная вершина ребра e в обобщенном графе зависимостей (номер инструкции $\sigma(e)$ начиная с нуля).
3. Конечная вершина ребра e в обобщенном графе зависимостей (номер инструкции $\delta(e)$ начиная с нуля).
4. Глубина зависимости ($1 + p_e$ в смысле определения (1.1)).
5. Номер доступа к массиву в инструкции $\sigma(e)$ начиная с нуля.
6. Номер доступа к массиву в инструкции $\delta(e)$ начиная с нуля.

7. Отношение общего вида, представляющее многогранник зависимости R_e.

Пример описания зависимости по данным $S_1 \rightarrow S_2$ для доступов `A[l][k]` и `A[i][k]` в программе, приведенной на листинге 2.1:

```
# Description of dependence 1
# type
RAW #(flow)
# From source statement id
0
# To target statement id
1
# Depth
1
# From source access ref
1
# To target access ref
2
# Dependence domain
UNDEFINED
24 14 5 6 0 1
# e/i| c1   c2   c3   c4   c5 | i1   i2   i3   i4   i5   i6 | P1 |  1
   1    1    0    0    0    0    0    0    0    0    0    0    0    0    ## c1 >= 0
   1   -1    0    0    0    0    0    0    0    0    0    0    1   -1    ## -c1+P1-1 >= 0
   1    0    0    0    0    0    0    0    0    0    0    0    1   -1    ## P1-1 >= 0
   1   -1    1    0    0    0    0    0    0    0    0    0    0   -1    ## -c1+c2-1 >= 0
   1    0   -1    0    0    0    0    0    0    0    0    0    1   -1    ## -c2+P1-1 >= 0
   1   -1    0    0    0    0    0    0    0    0    0    0    1   -2    ## -c1+P1-2 >= 0
   1    0    0    0    0    0    1    0    0    0    0    0    0    0    ## i1 >= 0
   1    0    0    0    0    0   -1    0    0    0    0    0    1   -1    ## -i1+P1-1 >= 0
   1    0    0    0    0    0    0    0    0    0    0    0    1   -1    ## P1-1 >= 0
   1    0    0    0    0    0   -1    1    0    0    0    0    0   -1    ## -i1+i2-1 >= 0
   1    0    0    0    0    0    0   -1    0    0    0    0    1   -1    ## -i2+P1-1 >= 0
   1    0    0    0    0    0   -1    0    0    0    0    0    1   -2    ## -i1+P1-2 >= 0
   1    0    0    0    0    0   -1    0    1    0    0    0    0   -1    ## -i1+i3-1 >= 0
   1    0    0    0    0    0    0    0   -1    0    0    0    1   -1    ## -i3+P1-1 >= 0
   0    0    0   -1    0    0    0    0    0    0    0    0    0    4    ## c3 == 4
   0    0    1    0   -1    0    0    0    0    0    0    0    0    0    ## c2-c4 == 0
   0    1    0    0    0   -1    0    0    0    0    0    0    0    0    ## c1-c5 == 0
   0    0    0    0    0    0    0    0    0    1    0    0    0   -4    ## i4-4 == 0
   0    0    0    0    0    0    0   -1    0    0    1    0    0    0    ## -i2+i5 == 0
   0    0    0    0    0    0   -1    0    0    0    0    1    0    0    ## -i1+i6 == 0
   0    0    0   -1    0    0    0    0    0    1    0    0    0    0    ## c3 == i4
   0    0    0    0   -1    0    0    0    0    0    1    0    0    0    ## c4 == i5
   0    0    0    0    0   -1    0    0    0    0    0    1    0    0    ## c5 == i6
   1   -1    0    0    0    0    1    0    0    0    0    0    0    0    ## -c1+i1 >= 0
```

Инструмент candl [36] генерирует отдельное описание зависимости для каждого возможного значения глубины, поэтому результат его работы может содержать различные описания для совпадающих инструкций и доступов к памяти, отличающиеся только указанной глубиной зависимости и последней секцией в матрице ограничений, фиксирующей лексикографический порядок зависимых операций.

В таком описании отсутствует явное представление h-преобразования, что затрудняет реализацию техники «lastwriter», которая заключается в удалении транзитивных зависимостей путем вычисления последнего конфликтующего доступа в случаях RAW/WAW. Исключение из рассмотрения таких зависимостей, которые присутствуют в транзитивном замыкании Γ, но отсутствуют в Γ, не нарушает выполнения принципа причинности, но упрощает анализ программы. Кроме того, устранение зависимостей типа WAR/WAW дает больше потенциальных возможностей для параллельного выполнения операций [117, с. 15]. Транслятор pluto использует библиотеку isl [49] вместо candl, если пользователь требует включения опции `--lastwriter`. В ilpy подобное поведение достигается предобработкой зависимостей, а именно секции лексикографического предшествования в матрице ограничений: если секция не пуста, и последнее условие является неравенством $p(x) \geqslant 0$, то оно заменяется на равенство $p(x) = 0$. Технически в последней строке матрицы ограничений значение в первом столбце изменяется с 1 на 0. Модифицированные многогранники зависимостей фигурируют в выходном файле `.ilpy`. Если многогранник зависимости остался непустым, значит он определяет последний конфликтующий доступ согласно лексикографическому порядку, и модификация корректна. В противном случае модификация некорректна, так как влечет потерю информации о зависимости.

3.3.3 Формирование ограничений для зависимостей по данным

Для линеаризации систем (2.6), (2.8) и (2.13) используется лемма Фаркаша и метод неопределенных коэффициентов. Для того, чтобы применить лемму Фаркаша, необходимо обеспечить в описании многогранников только условия в виде неравенств. Это легко сделать заменой каждого условия в виде равенства $p(x) = 0$ на два неравенства: $p(x) \geqslant 0$ и следующего за ним $-p(x) \geqslant 0$.

Матрица ограничений каждого многогранника зависимостей подвергается указанной трансформации, при этом первый столбец отбрасывается, так как все условия становятся единообразны. Удваивается количество условий в секциях 3, 4, 5 и 6 (последнее неравенство, если оно присутствовало, было заменено равенством на этапе предобработки). Порядок следования условий сохраняется.

Для удобства дальнейшего изложения введем обозначения для индексов первого и последнего условия в каждой секции матрицы ограничений многогранника зависимости:

1. $source^{start}_{domain}, \ldots, source^{end}_{domain}$;
2. $target^{start}_{domain}, \ldots, target^{end}_{domain}$;
3. $source^{start}_{access}, \ldots, source^{end}_{access}$;
4. $target^{start}_{access}, \ldots, target^{end}_{access}$;
5. $access^{start}_{eq}, \ldots, access^{end}_{eq}$;
6. $precedence^{start}, \ldots, precedence^{end}$.

Индексирование начинается с нуля, то есть $source^{start}_{domain} = 0$. Поскольку секции следуют друг за другом без зазоров в матрице ограничений, будем использовать введенные обозначения индексов для указания фрагментов матрицы, захватывающих более одной секции.

При нахождении расписаний и размещений вычислений фигурируют следующие условия, порождаемые каждой зависимостью по данным e:

– `LEGAL_SHEDULE`: $\varphi_{\delta(e)}(\vec{i}_{\delta(e)}, \vec{z}) - \varphi_{\sigma(e)}(\vec{i}_{\sigma(e)}, \vec{z}) \geqslant 1$. Одномерное расписание вычислений φ, не нарушающее принцип причинности, система (2.6).
– `FCO_PLACEMENT`: $\varphi_{\delta(e)}(\vec{i}_{\delta(e)}, \vec{z}) - \varphi_{\sigma(e)}(\vec{i}_{\sigma(e)}, \vec{z}) \geqslant 0$. Одномерное размещение вычислений φ, обладающее свойством вперед направленных коммуникаций, система (2.13).
– `LEGALITY_DECISION`: $\varphi_{\delta(e)}(\vec{i}_{\delta(e)}, \vec{z}) - \varphi_{\sigma(e)}(\vec{i}_{\sigma(e)}, \vec{z}) \geqslant \varepsilon_e$. Компонент φ многомерного расписания вычислений, не нарушающего принцип причинности, система (2.8).
– `DEP_LOWER_BOUND`: $\varphi_{\delta(e)}(\vec{i}_{\delta(e)}, \vec{z}) - \varphi_{\sigma(e)}(\vec{i}_{\sigma(e)}, \vec{z}) + \vec{l}_e \cdot \vec{z} + l^0_e \geqslant 0$. Одномерное размещение вычислений φ, не обладающее свойством вперед направленных коммуникаций. Замена условию `FCO_PLACEMENT`, когда оно невыполнимо, и разность аффинных отображений требуется ограничивать снизу.
– `DEP_UPPER_BOUND`: $-\varphi_{\delta(e)}(\vec{i}_{\delta(e)}, \vec{z}) + \varphi_{\sigma(e)}(\vec{i}_{\sigma(e)}, \vec{z}) + \vec{l}_e \cdot \vec{z} + l^0_e \geqslant 0$. Одномерное расписание или размещение вычислений, ограничение разности аффинных отображений сверху, система (2.6) и (2.13).

Каждое из этих условий преобразуется применением леммы Фаркаша (пример — система (2.7)), а затем метода неопределенных коэффициентов. Обозначим m_D матрицу ограничений, описывающую многогранник D, $\mu^{sign}_S \in \{-1, 1\}$ — знак аффинного отображения φ_S инструкции S в рассматриваемом усло-

вии. Условие DEP_UPPER_BOUND подразумевает $\mu^{sign}_{\sigma(e)} = 1$, $\mu^{sign}_{\delta(e)} = -1$. Для всех остальных условий знаки обратные. Приравнивание коэффициентов при индексах итерации, индексах массива, внешних переменных программы, а также констант, порождает новые условия в виде равенств для каждой зависимости по данным e. Зависимости по данным, как и инструкции, перебираются в порядке возрастания их номеров (определяется внутренними механизмами библиотеки osl [111]), и для каждой из них формируются 6 групп ограничений в виде равенств. Новые ограничения представляются в виде строк матрицы, имеющей структуру, представленную в таблице 2.

Таблица 2 — Структура матрицы для хранения ограничений в виде равенств

			$i = 1, \ldots, m$		$i = 1, \ldots, n$					
			$\mu_{S_i,k}$, $k = 1, \ldots, p_{D_{S_i}}$	$\mu_{S_i,0}$	$\lambda_{e_i,k}$, $k = 1, \ldots, p_{R_{e_i}}$	$\lambda_{e_i,0}$	$\lambda'_{e_i,k}$, $k = 1, \ldots, p_{R_{e_i}}$	$\lambda'_{e_i,0}$	$\vec{l}^{(k)}_{e_i}$, $k = 1, \ldots, q_z$	$l^0_{e_i}$
$j = 1, \ldots, n$	Для каждого условия	$\vec{i}^{(v)}_{\sigma(e_j)}$, $v = 1, \ldots, p_{\sigma(e_j)}$								
		$\vec{g}^{(v)}_{a_{\sigma(e_j)}}$, $v = 1, \ldots, p_A + 1$								
		$\vec{i}^{(v)}_{\delta(e_j)}$, $v = 1, \ldots, p_{\delta(e_j)}$								
		$\vec{g}^{(v)}_{a_{\delta(e_j)}}$, $v = 1, \ldots, p_A + 1$								
		$\vec{z}^{(v)}$, $v = 1, \ldots, q_z$								
		1								

Значение элемента матрицы (i,j) определяется коэффициентом при переменной, указанной в названии столбца j, в равенстве, определяемом строкой i (с учетом множителей μ^{sign}_S). Если переменная в равенстве не участвует, то значение соответствующего элемента матрицы считается нулевым, но в память не заносится, так как матрица хранится в формате списка координат, ориентированном на хранение разреженных матриц и поддерживаемом библиотекой glpk [115].

1. Для $\vec{i}^{(v)}_{\sigma(e)}$, $\quad v = 1, \ldots, p_{\sigma(e)}$:

$$\mu^{sign}_{\sigma(e)} \sum_{k=1}^{p_{D_{\sigma(e)}}} \mu_{\sigma(e),k} m_{D_{\sigma(e)}}[k-1][v] - \sum_{k=source^{start}_{domain}}^{source^{end}_{domain}} \lambda^*_{e,k+1} m_{R_e}[k][v-1] -$$

$$- \sum_{k=source^{start}_{access}}^{source^{end}_{access}} \lambda^*_{e,k+1} m_{R_e}[k][v-1] - \sum_{k=precedence^{start}}^{precedence^{end}} \lambda^*_{e,k+1} m_{R_e}[k][v-1] = 0.$$

2. Для $\vec{g}^{(v)}_{a_{\sigma(e)}}, \quad v = 1, \ldots, p_A + 1$:

$$- \sum_{k=source^{start}_{access}}^{source^{end}_{access}} \lambda^*_{e,k+1} m_{R_e}[k][v - 1 + p_{\sigma(e)}] -$$
$$- \sum_{k=access^{start}_{eq}}^{access^{end}_{eq}} \lambda^*_{e,k+1} m_{R_e}[k][v - 1 + p_{\sigma(e)}] = 0.$$

3. Для $\vec{i}^{(v)}_{\delta(e)}, \quad v = 1, \ldots, p_{\delta(e)}$:

$$\mu^{sign}_{\delta(e)} \sum_{k=1}^{p_{D_{\delta(e)}}} \mu_{\delta(e),k} m_{D_{\delta(e)}}[k-1][v] -$$
$$- \sum_{k=target^{start}_{domain}}^{target^{end}_{domain}} \lambda^*_{e,k+1} m_{R_e}[k][v - 1 + p_{\sigma(e)} + (p_A + 1)] -$$
$$- \sum_{k=target^{start}_{access}}^{target^{end}_{access}} \lambda^*_{e,k+1} m_{R_e}[k][v - 1 + p_{\sigma(e)} + (p_A + 1)] -$$
$$- \sum_{k=precedence^{start}}^{precedence^{end}} \lambda^*_{e,k+1} m_{R_e}[k][v - 1 + p_{\sigma(e)} + (p_A + 1)] = 0.$$

4. Для $\vec{g}^{(v)}_{a_{\delta(e)}}, \quad v = 1, \ldots, p_A + 1$:

$$- \sum_{k=target^{start}_{access}}^{target^{end}_{access}} \lambda^*_{e,k+1} m_{R_e}[k][v - 1 + p_{\sigma(e)} + (p_A + 1) + p_{\delta(e)}] -$$
$$- \sum_{k=access^{start}_{eq}}^{access^{end}_{eq}} \lambda^*_{e,k+1} m_{R_e}[k][v - 1 + p_{\sigma(e)} + (p_A + 1) + p_{\delta(e)}] = 0.$$

5. Для $\vec{z}^{(v)}, \quad v = 1, \ldots, q_z$:

$$\mu^{sign}_{\sigma(e)} \sum_{k=1}^{p_{D_{\sigma(e)}}} \mu_{\sigma(e),k} m_{D_{\sigma(e)}}[k-1][v + p_{\sigma(e)}] +$$

$$+ \mu^{sign}_{\delta(e)} \sum_{k=1}^{p_{D_{\delta(e)}}} \mu_{\delta(e),k} m_{D_{\delta(e)}}[k-1][v + p_{\delta(e)}] -$$

$$- \sum_{k=source^{start}_{domain}}^{target^{end}_{access}} \lambda^*_{e,k+1} m_{R_e}[k][v - 1 + p_{\sigma(e)} + 2(p_A + 1) + p_{\delta(e)}] +$$

$$+ \begin{cases} \vec{l}_e^{(v)} & \text{для DEP\_LOWER\_BOUND и DEP\_UPPER\_BOUND} \\ 0 & \text{иначе} \end{cases} = 0.$$

6. Для констант:

$$\mu^{sign}_{\sigma(e)} \left(\mu_{\sigma(e),0} + \sum_{k=1}^{p_{D_{\sigma(e)}}} \mu_{\sigma(e),k} m_{D_{\sigma(e)}}[k-1][p_{\sigma(e)} + q_z + 1] \right) +$$

$$+ \mu^{sign}_{\delta(e)} \left(\mu_{\delta(e),0} + \sum_{k=1}^{p_{D_{\delta(e)}}} \mu_{\delta(e),k} m_{D_{\delta(e)}}[k-1][p_{\delta(e)} + q_z + 1] \right) - \lambda^*_{e,0} -$$

$$- \sum_{k=source^{start}_{domain}}^{target^{end}_{access}} \lambda^*_{e,k+1} m_{R_e}[k][p_{\sigma(e)} + 2(p_A + 1) + p_{\delta(e)} + q_z] -$$

$$- \sum_{k=precedence^{start}}^{precedence^{end}} \lambda^*_{e,k+1} m_{R_e}[k][p_{\sigma(e)} + 2(p_A + 1) + p_{\delta(e)} + q_z] +$$

$$+ \begin{cases} l^0_e & \text{для DEP\_LOWER\_BOUND и DEP\_UPPER\_BOUND} \\ -l^0_e & \text{для LEGALITY\_DECISION в роли } \varepsilon_e \\ 0 & \text{иначе} \end{cases} =$$

$$= \begin{cases} 1 & \text{для LEGAL\_SCHEDULE} \\ 0 & \text{иначе} \end{cases}.$$

При обработке параметров и констант первые две суммы рассматриваются только если e не является петлей в обобщенном графе зависимостей. В противном случае они взаимно уничтожаются, так как $\mu^{sign}_{\sigma(e)} = -\mu^{sign}_{\delta(e)}$. Для DEP_UPPER_BOUND λ^* соответствует λ', во всех остальных случаях — λ.

3.3.4 Формирование ограничений для доступов к массивам

Для линеаризации системы (2.16) также используется лемма Фаркаша и метод неопределенных коэффициентов. При совместном нахождении размещений вычислений и данных фигурируют следующие условия, порождаемые каждым доступом к данным a:

- `FCO_PLACEMENT`: $\pi_{S_a}(\vec{i}_{S_a}, \vec{z}) - \eta_{A_a}(g_a(\vec{i}_{S_a}), \vec{z}) \geqslant 0$, если a — чтение; $\eta_{A_a}(g_a(\vec{i}_{S_a}), \vec{z}) - \pi_{S_a}(\vec{i}_{S_a}, \vec{z}) \geqslant 0$, если a — запись. Одномерное размещение вычислений π и данных η, обладающее свойством вперед направленных коммуникаций.
- `DEP_UPPER_BOUND`: $-\pi_{S_a}(\vec{i}_{S_a}, \vec{z}) + \eta_{A_a}(g_a(\vec{i}_{S_a}), \vec{z}) + \vec{l}_a \cdot \vec{z} + l_a^0 \geqslant 0$, если a — чтение; $-\eta_{A_a}(g_a(\vec{i}_{S_a}), \vec{z}) + \pi_{S_a}(\vec{i}_{S_a}, \vec{z}) + \vec{l}_a \cdot \vec{z} + l_a^0 \geqslant 0$, если a — запись. Одномерное размещение вычислений π и данных η, ограничение разности ν аффинных отображений сверху.
- `ACCESS_LOWER_BOUND`: $\pi_{S_a}(\vec{i}_{S_a}, \vec{z}) - \eta_{A_a}(g_a(\vec{i}_{S_a}), \vec{z}) + \vec{l}_a \cdot \vec{z} + l_a^0 \geqslant 0$; `ACCESS_UPPER_BOUND`: $-\pi_{S_a}(\vec{i}_{S_a}, \vec{z}) + \eta_{A_a}(g_a(\vec{i}_{S_a}), \vec{z}) + \vec{l}_a \cdot \vec{z} + l_a^0 \geqslant 0$. Одномерное размещение вычислений π и данных η, не обладающее свойством вперед направленных коммуникаций. Замена паре условий `FCO_PLACEMENT` и `DEP_UPPER_BOUND`, когда `FCO_PLACEMENT` не невыполнимо, и разность аффинных отображений требуется ограничивать снизу и сверху.
- `ACCESS_NONNEGATIVE`: $\vec{v}_{A_a} \cdot \vec{g} + \vec{v}'_{A_a} \cdot \vec{z} + v^0_{A_a} \geqslant 0$, $\vec{g} \in G_a$. Одномерное размещение данных η: общий вид и фиксация неотрицательности для элементов массива A_a с индексом $\vec{g}$, к которым осуществляется доступ в a.

Каждое из этих условий преобразуется применением леммы Фаркаша, а затем метода неопределенных коэффициентов. В выражении (2.18) возможна прямая подстановка $g_{a_j}(\vec{i}_{S_{a_j}})$ в неравенства, описывающие многогранник G_{a_j}. После приведения подобных слагаемых могут присутствовать артефакты в виде совпадающих неравенств, например, если индексы массива по некоторым измерениям совпадают. Исключить подобные артефакты, и получить многогранник в упрощенной форме можно следующим техническим решением, опирающимся на возможности библиотеки polylib [108]: если G_{a_j} получен как `Polyhedron_Image`$(D_{S_{a_j}}, g_{a_j})$, то искомый многогранник G'_{a_j} получается как прообраз G_{a_j} по функции g_{a_j}: `Polyhedron_Preimage`(G_{a_j}, g_{a_j}). Количество

ограничений, описывающих $G^{'}_{a_j}$ может быть меньше, чем для G_{a_j}. Количество столбцов в матрице ограничений совпадает у $D_{S_{a_j}}$ и $G^{'}_{a_j}$. Измерениям $D_{S_{a_j}}$, не участвующим в описании $G^{'}_{a_j}$, соответствуют столбцы матрицы ограничений, заполненные нулями. Применяя лемму Фаркаша с учетом описания многогранника $G^{'}_{a_j}$, получаем:

$$\eta_{A_{a_j}}(g_{a_j}(\vec{i}_{S_{a_j}}), \vec{z}) = \mu^{'}_{a_j,0} + \sum_{k=1}^{p_{G^{'}_{a_j}}} \mu^{'}_{a_j,k} \left(\vec{a}^{'}_{a_j,k} \cdot \begin{bmatrix} \vec{i}_{S_{a_j}} \\ \vec{z} \end{bmatrix} + b^{'}_{a_j,k} \right), \quad j = 1, \ldots, k, \quad (3.6)$$

где $\mu^{'}_{a_j,*} \geqslant 0$ — множители Фаркаша, а $p_{G^{'}_{a_j}}$ неравенств $\vec{a}^{'}_{a_j,k} \cdot \begin{bmatrix} \vec{i}_{S_{a_j}} \\ \vec{z} \end{bmatrix} + b^{'}_{a_j,k} \geqslant 0$ определяют прообраз $G^{'}_{a_j}$.

Обозначим $\mu^{sign}_{\pi} \in \{-1, 1\}$ — знак аффинного отображения π_{S_a} инструкции S_a в рассматриваемом условии, и $\mu^{sign}_{\eta} \in \{-1, 1\}$ — знак аффинного отображения η_{A_a} для массива A_a в рассматриваемом условии. Приравнивание коэффициентов при индексах итерации $\vec{i}_{S_a}$, внешних переменных программы, а также констант, порождает новые условия в виде равенств для каждого доступа к данным a. Доступы к данным, как инструкции и массивы, перебираются в порядке возрастания их номеров (определяется внутренними механизмами библиотеки osl [111]), и для каждого из них формируется 3 группы ограничений в виде равенств. Новые ограничения представляются в виде строк матрицы, имеющей структуру, представленную в таблице 3. В целях упрощения описания структуры столбцов матрицы таблица изложена в транспонированном виде.

Значение элемента матрицы (i,j) определяется коэффициентом при переменной, указанной в названии столбца j, в равенстве, определяемом строкой i (с учетом множителей μ^{sign}_{π} и μ^{sign}_{η}). Если переменная в равенстве не участвует, то значение соответствующего элемента матрицы считается нулевым, но в память не заносится, так как матрица хранится в формате списка координат, ориентированном на хранение разреженных матриц и поддерживаемом библиотекой glpk [115].

Для условий `FCO_PLACEMENT`, `DEP_UPPER_BOUND`, `ACCESS_LOWER_BOUND`, `ACCESS_UPPER_BOUND` формируются следующие 3 группы ограничений в виде равенств:

Таблица 3 — Структура матрицы для хранения ограничений в виде равенств (транспонированный вид)

		$j = 1, \ldots, k$		
		Для каждого условия		
		$\vec{i}^{(v)}_{S_{a_j}}, v = 1, \ldots, p_{S_{a_j}}$	$\vec{z}^{(v)}, v = 1, \ldots, q_z$	1
$i = 1, \ldots, m$	$\mu_{S_i,k}, k = 1, \ldots, p_{D_{S_i}}$			
	$\mu_{S_i,0}$			
$i = 1, \ldots, k$	$\mu'_{a_i,k}, k = 1, \ldots, p_{G'_{a_i}}$			
	$\mu'_{a_i,0}$			
	$\lambda_{a_i,k}, k = 1, \ldots, p_{D_{S_{a_i}}}$			
	$\lambda_{a_i,0}$			
	$\lambda'_{a_i,k}, k = 1, \ldots, p_{D_{S_{a_i}}}$			
	$\lambda'_{a_i,0}$			
	$\vec{l}^{(k)}_{a_i}, k = 1, \ldots, q_z$			
	$l^0_{a_i}$			
$i = 1, \ldots, l$	$\vec{v}^{(k)}_{A_i}, k = 1, \ldots, p_{A_i}$			
	$\vec{v}'^{(k)}_{A_i}, k = 1, \ldots, q_z$			
	$v^0_{A_i}$			

1. Для $\vec{i}^{(v)}_{S_a}, \quad v = 1, \ldots, p_{S_a}$:

$$\mu_\pi^{sign} \sum_{k=1}^{p_{D_{S_a}}} \mu_{S_a,k} m_{D_{S_a}}[k-1][v] + \mu_\eta^{sign} \sum_{k=1}^{p_{G'_{S_a}}} \mu'_{a,k} m_{G'_{S_a}}[k-1][v] - \\ - \sum_{k=1}^{p_{D_{S_a}}} \lambda^*_{a,k} m_{D_{S_a}}[k-1][v] = 0.$$

2. Для $\vec{z}^{(v)}, \quad v = 1, \ldots, q_z$:

$$\mu_\pi^{sign} \sum_{k=1}^{p_{D_{S_a}}} \mu_{S_a,k} m_{D_{S_a}}[k-1][v+p_{S_a}] + \mu_\eta^{sign} \sum_{k=1}^{p_{G'_{S_a}}} \mu'_{a,k} m_{G'_{S_a}}[k-1][v+p_{S_a}] - \\ - \sum_{k=1}^{p_{D_{S_a}}} \lambda^*_{a,k} m_{D_{S_a}}[k-1][v+p_{S_a}] + \\ + \begin{cases} \vec{l}^{(v)}_a & \text{для DEP\_UPPER\_BOUND,} \\ & \text{ACCESS\_LOWER\_BOUND и ACCESS\_UPPER\_BOUND} \\ 0 & \text{иначе} \end{cases} = 0.$$

3. Для констант:

$$\mu_{\pi}^{sign}\left(\mu_{S_a,0}+\sum_{k=1}^{p_{D_{S_a}}}\mu_{S_a,k}m_{D_{S_a}}[k-1][1+p_{S_a}+q_z]\right)+$$
$$+\mu_{\eta}^{sign}\left(\mu_{a,0}^{'}+\sum_{k=1}^{p_{G'_{S_a}}}\mu_{a,k}^{'}m_{G'_{S_a}}[k-1][1+p_{S_a}+q_z]\right)-$$
$$-\left(\lambda_{a,0}^{*}+\sum_{k=1}^{p_{D_{S_a}}}\lambda_{a,k}^{*}m_{D_{S_a}}[k-1][1+p_{S_a}+q_z]\right)+$$
$$+\begin{cases} l_a^0 & \text{для DEP\_UPPER\_BOUND,} \\ & \text{ACCESS\_LOWER\_BOUND и ACCESS\_UPPER\_BOUND} = 0. \\ 0 & \text{иначе} \end{cases}$$

Для DEP_UPPER_BOUND и ACCESS_UPPER_BOUND λ^* соответствует $\lambda^{'}$, а для FCO_PLACEMENT и ACCESS_LOWER_BOUND — λ.

Для пары условий ACCESS_LOWER_BOUND и ACCESS_UPPER_BOUND используется один набор переменных-ограничителей $\vec{l_a}$ и l_a^0 в целях фиксации одинаковых требований к расстоянию использования данных независимо от направления коммуникации. Использование разных наборов переменных-ограничителей для реализации пары условий может обеспечить лучшую пространственную локальность использования данных ценой увеличения количества переменных в формулировке задачи ЛЦП.

Отдельно рассмотрим условие ACCESS_NONNEGATIVE. Для него также применяется лемма Фаркаша с учетом (3.6) и выполняется подстановка $g_a(\vec{i}_{S_a})$ вместо $\vec{g}$. Обозначим m_{g_a} матрицу ограничений, описывающую индексную функцию доступа g_a. Формируются следующие 3 группы ограничений в виде равенств:

1. Для $\vec{i}_{S_a}^{(v)}, \quad v = 1, \ldots, p_{S_a}$:

$$\sum_{k=1}^{p_{G'_{S_a}}}\mu_{a,k}^{'}m_{G'_{S_a}}[k-1][v]-\sum_{k=1}^{p_{A_a}}\vec{v}_{A_a}^{(k)}m_{g_a}[k][v+p_{A_a}]=0.$$

2. Для $\vec{z}^{(v)}, \quad v = 1, \ldots, q_z$:

$$\sum_{k=1}^{p_{G'_{S_a}}}\mu_{a,k}^{'}m_{G'_{S_a}}[k-1][v+p_{S_a}]-\sum_{k=1}^{p_{A_a}}\vec{v}_{A_a}^{(k)}m_{g_a}[k][v+p_{A_a}+p_{S_a}]-\vec{v}_{A_a}^{'(v)}=0.$$

3. Для констант:

$$\mu'_{a,0} + \sum_{k=1}^{p_{G'_{Sa}}} \mu'_{a,k} m_{G'_{Sa}}[k-1][1 + p_{S_a} + q_z] -$$
$$-v^0_{A_a} - \sum_{k=1}^{p_{A_a}} \vec{v}^{(k)}_{A_a} m_{g_a}[k][1 + p_{A_a} + p_{S_a} + q_z] = 0.$$

3.3.5 Нахождение расписания вычислений

Рассмотрим частный случай, когда обобщенный граф зависимостей программы не содержит циклов, при этом могут присутствовать петли. Рассмотрим вначале подграф, полученный исключением ребер, индуцирующих петли. В отсутствие петель рассматриваемый подграф принимает вид ориентированной сети. Все вершины, принадлежащие одному уровню сети, соответствуют инструкциям, динамические экземпляры которых могут быть выполнены параллельно, то есть в один момент логического времени. Таким образом, нахождение расписания вычислений для сети может быть сведено к топологической сортировке вершин, а точнее, к вычислению порядковой функции сети. Алгоритм Демукрона [118, с. 349], в основе которого лежит идея «послойного» удаления на каждой итерации тех вершин сети, что имеют нулевую степень захода, может быть применен для определения уровней вершин, и, следовательно, нахождения расписания вычислений.

Если обобщенный граф зависимостей программы не содержит петель и циклов, то достаточно одномерного расписания вычислений: $\vec{\theta}_{S_i} = [ord(S_i)], \quad i = 1, \ldots, m$, где $ord\colon \{S_i | i = 1, \ldots, m\} \to \mathbb{N}_0$ — порядковая функция сети. Если присутствуют петли, то первый компонент многомерного расписания может быть вычислен на основе порядковой функции сети для подграфа без петель, и будет удовлетворять те зависимости по данным, которые существуют между различными инструкциями. Оставшиеся зависимости, которые индуцируют петли, могут быть удовлетворены следующими компонентами многомерного расписания, для вычисления которых можно применить разработанный метод на основе жадной схемы Футриера, оперирующий оптимальным набором аффинных отображений на каждой итерации.

Нахождение расписания вычислений в ilpy для обобщенного графа зависимостей E происходит по следующему алгоритму:

1. Если задан параметр `--smdp`, то
   а) $E'_{net} \leftarrow \{e|e \in E \wedge \sigma(e) \neq \delta(e)\}$; $E'_{loop} \leftarrow \{e|e \in E \wedge \sigma(e) = \delta(e)\}$.
   б) Выполнить алгоритм Демукрона для графа $\langle\{S_i|i = 1,\ldots,m\}; E'_{net}\rangle$ и получить порядковую функцию сети ord. Если граф не является сетью, транслятор прерывает работу: об этом сигнализирует ситуация, если на очередной итерации алгоритма Демукрона не обнаружилось вершин с нулевой полустепенью захода, и при этом остались вершины, нераспределенные по уровням [118, с. 354].
   в) $\vec{\theta}^{(1)}_{S_i} \leftarrow ord(S_i), \quad i = 1,\ldots,m$.
   г) Если $E'_{loop} \neq \varnothing$, то выполнить алгоритм на основе жадной схемы Футриера для E'_{loop} начиная со второго компонента многомерного расписания.
2. Иначе выполнить алгоритм на основе жадной схемы Футриера для E начиная с первого компонента многомерного расписания.

Реализация алгоритма на основе жадной схемы Футриера предполагает конструирование матрицы ограничений для задач ЛЦП двух видов.

– Выяснение зависимостей, которые можно удовлетворить очередным компонентом многомерного расписания. При линеаризации системы (2.8) конструируются ограничения вида `LEGALITY_DECISION`. Для индикаторных переменных ε_e не конструируются отдельные ограничения, выполняется только вызов glpk [115] `glp_set_col_bnds(…, GLP_DB, 0, 1)` для фиксации $\varepsilon_e \in \{0,1\}$.
– Вычисление очередного компонента многомерного расписания. При линеаризации системы (2.6) конструируются ограничения вида `LEGAL_SHEDULE` и `DEP_UPPER_BOUND`.

Для каждого ограничения в виде равенства выполняется вызов `glp_set_row_bnds(…, GLP_FX, fv, fv)`, где `fv` представляет константу в правой части ограничения. Для всех столбцов, кроме соответствующих индикаторным переменным, фиксируется неотрицательность: `glp_set_row_bnds(…, GLP_LO, 0, 0)`. Коэффициенты при переменных в целевой функции задаются вызовом `glp_set_obj_coef` в соответствии с (2.9) и (2.5), при этом вместо $\vec{z}$

используются веса внешних переменных программы, которые можно передавать в параметрах командной строки.

Решение задачи ЛЦП считается успешным при выполнении следующих условий:

1. `glp_simplex(lp, NULL) == 0` — решатель задачи ЛП (симплекс-метод) завершил работу успешно;
2. `glp_get_status(lp) == GLP_OPT` — решатель задачи ЛП нашел оптимум;
3. `glp_intopt(lp, NULL) == 0` — решатель задачи ЛЦП (метод ветвей и границ) завершил работу успешно;
4. `glp_mip_status(lp) == GLP_OPT` — решатель задачи ЛЦП нашел оптимум.

В противном случае транслятор прерывает работу.

На основе найденных множителей Фаркаша вычисляются аффинные отображения инструкций, и выводится диагностическая информация о каждой аффинной функции L_e^{τ}.

3.3.6 Нахождение размещения вычислений и данных

Нахождение пространственных отображений при распараллеливании линейной программы предполагает конструирование матрицы ограничений для задач ЛЦП двух видов.

– Нахождение одномерного размещения вычислений. При линеаризации системы (2.13) конструируются ограничения вида `FCO_PLACEMENT` и `DEP_UPPER_BOUND`, если фиксируется свойство вперед направленных коммуникаций. В противном случае применяется `DEP_LOWER_BOUND` вместо `FCO_PLACEMENT`.
– Совместное нахождение одномерного размещения вычислений и данных. При линеаризации системы (2.16) конструируются ограничения вида `FCO_PLACEMENT` и `DEP_UPPER_BOUND`, если фиксируется свойство вперед направленных коммуникаций. В противном случае вместо них применяется пара `ACCESS_LOWER_BOUND` и `ACCESS_UPPER_BOUND`.

Исключение нулевого решения и фиксация линейной независимости размещения вычислений от компонентов ранее найденного многомерного расписания основываются на конструировании ограничений для условий вида $\vec{x}_{S_i} \neq \vec{0}, \quad i = 1, \ldots, m$, где каждый компонент вектора $\vec{x}_{S_i}$ является взвешенной суммой множителей Фаркаша: $\vec{x}_{S_i}^{(j)} = \sum\limits_{k=1}^{p_{D_{S_i}}} \mu_{S_i,k} m_{\vec{x}_{S_i}}[j-1][k-1], \quad j = 1, \ldots, p$, где $m_{\vec{x}_{S_i}}$ — матрица коэффициентов размера $p \times p_{D_{S_i}}$. Пусть вектор $\vec{y}$ имеет p компонентов, как и вектор $\vec{x}_{S_i}$. Рассмотрим скалярное произведение:

$$\begin{aligned} \vec{y} \cdot \vec{x}_{S_i} = \sum_{j=1}^{p} \vec{y}^{(j)} \vec{x}_{S_i}^{(j)} = \sum_{j=1}^{p} \left(\vec{y}^{(j)} \sum_{k=1}^{p_{D_{S_i}}} \mu_{S_i,k} m_{\vec{x}_{S_i}}[j-1][k-1] \right) = \\ = \sum_{j=1}^{p} \sum_{k=1}^{p_{D_{S_i}}} \vec{y}^{(j)} \mu_{S_i,k} m_{\vec{x}_{S_i}}[j-1][k-1] = \\ = \sum_{k=1}^{p_{D_{S_i}}} \left(\mu_{S_i,k} \sum_{j=1}^{p} \vec{y}^{(j)} m_{\vec{x}_{S_i}}[j-1][k-1] \right). \end{aligned} \tag{3.7}$$

Тогда условие $\vec{x}_{S_i} \neq \vec{0}$ описывается следующими ограничениями согласно (2.10):

1. $-b + 1 \leqslant \sum\limits_{k=1}^{p_{D_{S_i}}} \mu_{S_i,k} m_{\vec{x}_{S_i}}[j-1][k-1] \leqslant b - 1, \quad j = 1, \ldots, p.$
2. $\sum\limits_{k=1}^{p_{D_{S_i}}} \left(\mu_{S_i,k} \sum\limits_{j=1}^{p} b^{j-1} m_{\vec{x}_{S_i}}[j-1][k-1] \right) + \varepsilon b^{p+1} \geqslant 1.$
3. $-\sum\limits_{k=1}^{p_{D_{S_i}}} \left(\mu_{S_i,k} \sum\limits_{j=1}^{p} b^{j-1} m_{\vec{x}_{S_i}}[j-1][k-1] \right) - \varepsilon b^{p+1} \geqslant 1 - b^{p+1}.$

Значение параметра b выбрано равным 5 согласно рекомендациям, приведенным в [109]. В левой части второго и третьего ограничения-неравенства скалярное произведение вида (3.7) раскрыто для $\vec{y}$: $\vec{y}^{(j)} = b^{j-1}, \quad j = 1, \ldots, p$.

Для каждого ограничения-неравенства добавляется строка в матрицу ограничений задачи ЛЦП. Значение элемента j этой строки определяется коэффициентом при переменной в левой части неравенства, указанной в названии столбца j матрицы ограничений (таблицы 2 и 3). Для второго и третьего ограничения-неравенства матрица ограничений задачи ЛЦП дополняется столбцом для индикаторной переменной $\varepsilon \in \{0, 1\}$. Чтобы задать семантику неравенства для строки матрицы ограничений задачи ЛЦП, для первого двойного неравенства может быть использован вызов glpk [115] `glp_set_row_bnds(…,`

`GLP_DB, -b + 1, b - 1`), для второго — `glp_set_row_bnds(…, GLP_LO, 1, 0)`, для третьего — `glp_set_row_bnds(…, GLP_LO, 1 - bp, 0)`, где `bp` вычислено как b^{p+1}.

Ограничения-неравенства для условия $\vec{v}_{S_i} \neq \vec{0}$ могут быть сконструированы непосредственно, если задать $m_{\vec{v}_{S_i}}[r][c] = m_{D_{S_i}}[c][r+1], \quad r = 0, \ldots, p_{S_i} - 1, c = 0, \ldots, p_{D_{S_i}} - 1$.

Применим (3.7) для $\vec{y}$: $\vec{y}^T = \boldsymbol{H}^{\perp}_{S_i}[r], \quad r \in [1..p_{S_i}]$ и $\vec{x}_{S_i} = \vec{v}_{S_i}$:

$$\boldsymbol{H}^{\perp}_{S_i}[r]\vec{v}_{S_i} = \sum_{k=1}^{p_{D_{S_i}}} \left(\mu_{S_i,k} \sum_{j=1}^{p_{S_i}} \boldsymbol{H}^{\perp}_{S_i}[r][j] m_{\vec{v}_{S_i}}[j-1][k-1] \right) =$$

$$= \sum_{k=1}^{p_{D_{S_i}}} \left(\mu_{S_i,k} \sum_{j=1}^{p_{S_i}} \boldsymbol{H}^{\perp}_{S_i}[r][j] m_{D_{S_i}}[k-1][j] \right).$$

Пусть $\vec{x}_{S_i} = \boldsymbol{H}^{\perp}_{S_i}\vec{v}_{S_i}$, тогда матрица коэффициентов задается следующим образом: $m_{\vec{x}_{S_i}}[r-1][k-1] = \sum\limits_{j=1}^{p_{S_i}} \boldsymbol{H}^{\perp}_{S_i}[r][j] m_{D_{S_i}}[k-1][j], \quad r = 1, \ldots, p_{S_i}, k = 1, \ldots, p_{D_{S_i}}$. Конструирование ограничений-неравенств для фиксации линейной независимости размещения вычислений от компонентов ранее найденного многомерного расписания сводится к конструированию ограничений для $\vec{x}_{S_i} \neq \vec{0}$ с учетом (2.12).

Для каждого ограничения в виде равенства выполняется вызов `glp_set_row_bnds(…, GLP_FX, fv, fv)`, где `fv` представляет константу в правой части ограничения. Для всех столбцов, кроме соответствующих индикаторным переменным, по умолчанию фиксируется неотрицательность: `glp_set_row_bnds(…, GLP_LO, 0, 0)`. Если же пользователь задал параметр командной строки `--ilp-col-ub`, то вступает в силу еще одно ограничение сверху: `glp_set_row_bnds(…, GLP_DB, 0, ilp_col_ub)`. Коэффициенты при переменных в целевой функции задаются вызовом `glp_set_obj_coef` в соответствии с (2.5) и (2.15), при этом вместо $\vec{z}$ используются веса внешних переменных программы, которые можно передавать в параметрах командной строки.

Решение задачи ЛЦП считается успешным при выполнении следующих условий:

1. `glp_simplex(lp, NULL) == 0` — решатель задачи ЛП (симплекс-метод) завершил работу успешно;
2. `glp_get_status(lp) == GLP_OPT` — решатель задачи ЛП нашел оптимум;

3. `glp_intopt(lp, NULL) == 0` или `glp_intopt(lp, NULL) == GLP_ETMLIM` — решатель задачи ЛЦП (метод ветвей и границ) завершил работу успешно или прервал вычисления по достижению заданного пользователем лимита времени;
4. `glp_mip_status(lp) == GLP_OPT` или `glp_mip_status(lp) == GLP_FEAS` — решатель задачи ЛЦП нашел оптимум или остановился на допустимом субоптимальном решении.

В противном случае нахождение одномерного размещения вычислений (и данных) считается проваленным.

На основе найденных множителей Фаркаша вычисляются аффинные отображения инструкций, и выводится диагностическая информация о каждой аффинной функции L^{ρ}.

При построении матрицы $\boldsymbol{H}_{S_i}, \quad i = 1, \ldots, m$ учитываются не только компоненты многомерного расписания, но и аффинные отображения, соответствующие размещению вычислений. Это позволяет генерировать многомерные аффинные размещения вычислений (и данных) подобно тому, как выполняется построение набора линейно независимых отображений в pluto [37], что легло в основу реализации опции `--async`. При этом в матрицу $\boldsymbol{H}_{S_i}$ не добавляются строки, полностью состоящие из нулей, и строки, совпадающие с предыдущими с точностью до постоянного множителя, поскольку они препятствуют вычислению $\boldsymbol{H}_{S_i}^{\perp}$. Если не удалось сформировать ни одной строки, то $\boldsymbol{H}_{S_i}^{\perp} = I_{p_{S_i}}$, что означает допустимость использования любых выходных измерений D_{S_i} для вычисления компонента многомерного размещения вычислений. Если $\boldsymbol{H}_{S_i}^{\perp}$ содержит только нули, то ограничение линейной независимости не накладывается в силу невозможности его выполнения.

Нахождение размещения вычислений (и данных) в ilpy для обобщенного графа зависимостей E (и множества доступов к памяти $\{a\}$) происходит по следующему алгоритму:

1. $d \leftarrow 1$.
2. Вычислить $\pi_{S_i}^{(d)}, i = 1, \ldots, m$ (и $\eta_{A_i}^{(d)}, i = 1, \ldots, l$) как оптимальный набор аффинных отображений для E (и $\{a\}$).
3. Если требуется одномерное размещение вычислений (и данных), то вернуть 1.
4. Если существует инструкция S, для которой найдено менее p_S аффинных отображений (с учетом компонентов многомерного расписания и много-

мерного размещения вычислений), то выполнить $d \leftarrow d + 1$ и перейти к шагу 2.

5. Иначе вернуть d.

Результатом работы алгоритма является размещение вычислений (и данных), а также количество найденных аффинных отображений.

Нахождение компонента многомерного размещения вычислений (и данных) осуществляется в несколько попыток с последовательным ослаблением ограничений:

1. линейная независимость: ДА, свойство FCO: ДА;
2. линейная независимость: ДА, свойство FCO: НЕТ;
3. линейная независимость: НЕТ, свойство FCO: ДА;
4. линейная независимость: НЕТ, свойство FCO: НЕТ.

3.3.7 Генерация программного кода

Найденные расписания и размещения вычислений фигурируют в представлении программы OpenSCOP в отношениях типа `SCATTERING`. Для каждой инструкции S исходное отношение заменяется на новое, генерируемое по схеме, проиллюстрированной в таблице 4.

Таблица 4 — Структура матрицы для хранения аффинных отображений

		Выходные измерения	Входные измерения			
		$j = 1, \ldots, \dim \theta_S + \dim \pi_S$	$j = 1, \ldots, p_S$	$j = 1, \ldots, q_z$		
		c_j	$\vec{i}_S^{(j)}$	$\vec{z}^{(j)}$	1	
По умолчанию	$\vec{0}$	$-I_{\dim \theta_S + \dim \pi_S}$	$\vec{v}^{(j)}_{\theta_S^{(i)}}$	$\vec{v}'^{(j)}_{\theta_S^{(i)}}$	$v^0_{\theta_S^{(i)}}$	$i = 1, \ldots, \dim \theta_S$
			$\vec{v}^{(j)}_{\pi_S^{(i)}}$	$\vec{v}'^{(j)}_{\pi_S^{(i)}}$	$v^0_{\pi_S^{(i)}}$	$i = 1, \ldots, \dim \pi_S$
Если задан параметр `--async`	$\vec{0}$	$-I_{\dim \theta_S + \dim \pi_S}$	$\vec{v}^{(j)}_{\pi_S^{(i)}}$	$\vec{v}'^{(j)}_{\pi_S^{(i)}}$	$v^0_{\pi_S^{(i)}}$	$i = 1, \ldots, \dim \pi_S$
			$\vec{v}^{(j)}_{\theta_S^{(i)}}$	$\vec{v}'^{(j)}_{\theta_S^{(i)}}$	$v^0_{\theta_S^{(i)}}$	$i = 1, \ldots, \dim \theta_S$

Все ограничения являются равенствами. Если задан параметр `--async`, то генерируется (возможно, некорректная) программа с асинхронным параллелизмом: аффинные отображения, соответствующие размещению вычислений, выставляются перед аффинными отображениями, соответствующими расписанию вычислений. По умолчанию генерируется корректная программа с синхронным параллелизмом, где отображения, соответствующие размещению вычислений, следуют за отображениями, соответствующими расписанию вычислений.

Например, для параллельного варианта LU-разложения квадратной матрицы (листинг 2.3), формируются следующие два отношения для инструкций S_1 и S_2 соответственно:

```
# ============================================== Statement 1
...
SCATTERING
2 7 2 2 0 1
# e/i| c1   c2 |  k    l |  N |  1
   0   -1    0    2    0    0    0    ## c1 == 2*k
   0    0   -1    0    1    0   -1    ## c2 == l-1
# ============================================== Statement 2
...
SCATTERING
2 8 2 3 0 1
# e/i| c1   c2 |  k    i    j |  N |  1
   0   -1    0    2    0    0    0    1    ## c1 == 2*k+1
   0    0   -1    0    1    0    0   -1    ## c2 == i-1
```

Информация об именах новых индексных переменных c_j, $j = 1, \ldots, \dim \theta_S + \dim \pi_S$ содержится в отдельном расширении OpenSCOP scatnames. Для рассматриваемого примера LU-разложения (листинг 2.3) она имеет вид:

```
<scatnames>
t0 ilpp
</scatnames>
```

Для индексной переменной, соответствующей компоненту многомерного расписания $\theta_S^{(j)}$, $j = 1, \ldots, \dim \theta_S$, имя формируется по шаблону `t<j>`, если задан параметр `--smdp`, и `t<j-1>` по умолчанию.

Для индексной переменной, соответствующей компоненту многомерного размещения $\pi_S^{(j)}$, $j = 1, \ldots, \dim \pi_S$, имя формируется по шаблону `p<j-1>` для $j > 1$, и задается `ilpp` для $j = 1$. Таким образом, благодаря модификации cloog, первый компонент многомерного размещения всегда можно будет выявить в результирующей программе по индексной переменной `ilpp`, так как соответствующий цикл будет присутствовать в явном виде, даже если он содержит всего одну итерацию.

Итоговое представление OpenSCOP с модифицированными многогранниками зависимостей и найденными аффинными отображениями сохраняется в файле `.ilpy`. Генерация параллельной программы выполняется с помощью cloog [88], и результат сохраняется в файле `.cloog`. Параметры cloog «First Depth to Optimize Control» и «Last Depth to Optimize Control» могут быть изменены с помощью аргументов командной строки `--cloog-f` и `--cloog-l`.

3.4 Общие выводы по главе

Разработанный метод генерации параллельной MPI-программы позволяет организовать информационный обмен между параллельными процессами в случае явно заданного распределения данных между процессорами, основываясь на описании многогранников коммуникаций. Разработанная библиотека макросов blockdist.h позволяет обрабатывать многогранники коммуникаций в форме фрагментов линейных массивов, а также столбцов и строк матриц для избранных частных случаев размещения вычислений и данных. По сравнению с методом Дататри [90] исключается необходимость размещать входные данные на всех вычислительных устройствах.

Разработанное программное обеспечение — транслятор ilpy — может быть использовано для анализа потока данных в программах линейного класса, улучшения локальности использования данных при вычислениях, а также распараллеливания таких программ для универсальных многоядерных процессоров и построенных на их основе кластерных систем благодаря поддержке OpenMP и MPI в качестве технологий распараллеливания. В качестве направлений развития транслятора отмечаются решения следующих технических задач:

- поддержка отношений в виде объединения выпуклых многогранников в OpenSCOP;
- поддержка локальных измерений в многогранниках, описывающих домены, зависимости, доступы к данным, аффинные отображения;
- вычисление и применение h-преобразования для упрощения многогранников зависимостей;
- поддержка аффинных функций общего вида для размещения вычислений и данных в библиотеке макросов blockdist.h;
- разработка расширения OpenSCOP для многогранников коммуникаций и автоматизация расстановки макросов blockdist.h;
- оптимизация ветвлений машинным способом и подготовка параллельных циклов как расширение функциональности cloog.

Предметом дальнейших исследований являются техники передачи данных в информационных пакетах специального формата (например, разреженные матрицы), а также методы построения программ с асинхронным параллелизмом, позволяющие отказаться от барьерной синхронизации.

Глава 4. Экспериментальные исследования производительности параллельных программ

4.1 Тестовая среда: сборка и запуск приложений

Сборка всех тестовых программ производится с помощью утилиты GNU Make [119] в среде Linux. На листинге А.21 представлен общий файл сборки, применяемый для всех тестов с OpenMP. Вызов транслятора pluto выполняется с параметрами `--parallelize --lastwriter`, что подразумевает распараллеливание с OpenMP и рассмотрение при анализе зависимостей только последней операции, индуцирующей зависимость по данным для двух инструкций, что соответствует поведению ilpy. Итоговая компиляция выполняется с помощью gcc [120] с наивысшим уровнем оптимизации. В сборку статически включаются линейный конгруэнтный генератор [121] (приложение А.3), применяемый при заполнении массивов случайными данными, и различные утилиты для управления памятью и поддержки вывода статистики запуска (приложение А.4). Параллельные варианты программ, получаемых применением ilpy и pluto, проходят ручную постобработку перед включением в исходный текст теста. Имя файла определяется внешней переменной `alg`.

Сборка тестов с MPI включает большее количество этапов обработки исходного текста для pluto. На листинге А.27 представлен общий файл сборки, применяемый для всех тестов с MPI. Вызов транслятора pluto выполняется с параметрами `--distmem --commopt_fop --isldep --lastwriter --cloogsh --timereport`, применявшимися в публикации [90], но без разбиений (tiling), чтобы максимально приблизить поведение ilpy.

Предобработка исходного текста для транслятора pluto включает следующие шаги:

1. Включение заголовочных файлов (линейный конгруэнтный генератор `lcg.h`, утилиты `util.h`, макросы cloog `umacros.h`).
2. Декларация переменных для сбора статистики запуска: время исполнения `pluto_spent_time` и время информационного обмена с синхронизацией вычислений `mpi_spent_time`.

3. Декларация глобальных переменных, включающих размеры задачи и обрабатываемые массивы. Определяется внешней переменной `global_variables`.
4. Обрамление кода в функцию `$(alg)_pluto_mpi()`, которая будет вызываться из основного исходного текста теста. При этом применяются текстовые преобразования утилитой sed, определяемые внешней переменной `sed_preprocess_expr`.

Постобработка исходного текста, полученного на выходе транслятора pluto, включает следующие шаги:

1. Комментирование конструкций инициализации и завершения работы MPI, поскольку это выполняется в основном исходном тексте теста.
2. Удаление взаимодействия с файловой системой для поиска файлов-флагов, в тесте неиспользуемых. Это выполняется для исключения лишних задержек при измерении времени исполнения кода.
3. Вставка синхронизационного барьера перед началом и после окончания параллельного региона, время исполнения которого и представляет интерес во всех экспериментах.
4. Вставка конструкций для измерения времени исполнения (используется переменная pluto `t_local`) и времени информационного обмена и синхронизации вычислений (используется суммарное время, затраченное на упаковку, пересылку и распаковку пакетов `t_comm + t_pack + t_unpack`).

В силу подхода, реализованного pluto, не представляется возможным отдельно измерить время, затраченное на синхронизацию вычислений, поскольку синхронизационные барьеры встречаются неявно в применяемых функциях MPI, таких как `MPI_Alltoall`, или выставляются явно в виде `MPI_Waitall` для ожидания завершения информационного обмена.

Итоговая компиляция выполняется с помощью mpicc [122] с наивысшим уровнем оптимизации. В сборку статически включаются линейный конгруэнтный генератор [121] (приложение А.3), применяемый при заполнении массивов случайными данными, и различные утилиты для управления памятью и поддержки вывода статистики запуска (приложение А.4), а также функции поддержки исполнения pluto `polyrt.c`. Параллельные варианты программ, получаемые применением ilpy, проходят ручную постобработку перед включением в исходный

текст теста. Имя файла определяется внешней переменной `alg` и содержит суффикс `_mpi`.

Запуск всех тестовых приложений с MPI выполнялся с помощью менеджера ресурсов SLURM [123]. На листинге А.34 приведена функция, выполняющая запуск в эксклюзивном режиме для каждого узла, чтобы исключить влияние процессов других заданий на выполнение вычислений. Через переменную `nodes` передается количество узлов, а через переменную `tasks` — количество процессов на каждом узле. Дальнейшие действия не имеют смысл, если произведение двух переданных значений будет нулевым. Факт отправки задания фиксируется наличием файла журнала в файловой системе: если он присутствует, значит повторная отправка не производится, и выдается сообщение о наличии файла журнала. В противном случае выполняется постановка задания в очередь. Общее количество процессов и дополнительные параметры (такие, как применяемый параллельный вариант в переменной `method`, и прочие, переданные в функцию `submit`) передаются аргументами командной строки в скрипт запуска конкретного теста, имя которого определяется переменной `prog`.

Все тестовые запуски выполнялись на оборудовании РТУ МИРЭА. Была использована лезвийная система, включающая 8 идентичных лезвий (в дальнейшем каждое лезвие будет называться отдельной «машиной»). Каждое лезвие построено на базе процессора Intel(R) Xeon(R) CPU E5-2690 v2 @ 3.00GHz, имеет 16Гб оперативной памяти стандарта DDR3. Сеть передачи сообщений — InfiniBand с пропускной способностью 40Гбит/с. Операционная система — Linux CentOS 7.8 x64. Компилятор — gcc версии 4.8.5. Реализация MPI — openmpi-4.0.3rc4. Все лезвия организованы в кластер с помощью менеджера ресурсов SLURM и рассматриваются как единый ресурс. Менеджер ресурсов SLURM сконфигурирован на отдельной виртуальной машине за пределами кластера node1, которая исключается из рассмотрения при постановке вычислительных заданий в очередь. Запуск программ, распараллеленных с OpenMP, также осуществлялся с помощью SLURM: каждому тесту необходима только одна машина в эксклюзивном использовании (листинг А.33).

Далее будут рассмотрены пять программ из пакета polybench [110]. Каждый из пяти экспериментов по распараллеливанию будет включать:

– Рассмотрение исходного последовательного варианта программы, построение обобщенного графа зависимостей.

– Распараллеливание с применением разработанного транслятора ilpy и современного транслятора pluto: найденные пространственные и временные отображения, их свойства, результирующий исходный код.
– Графики ускорения вычислений и обсуждение вопросов, связанных с производительностью решений.

Все вычисления производятся с вещественными числами двойной точности (тип double). Сравнение времени выполнения программ ориентируется на среднее арифметическое значение, полученное для множества запусков: 10 для экспериментов с OpenMP и 100 для экспериментов с MPI, где выполнялись дополнительные измерения времени, затраченного на синхронизацию процессов и информационный обмен.

4.2 LU-разложение квадратной матрицы

4.2.1 Распараллеливание с OpenMP

Рассматривается программа из набора тестов polybench [110], выполняющая вычисления «на месте», то есть результат окажется в той же области памяти, что и входные данные (функция `lu_vanilla`, листинг Б.1). Разработанный транслятор ilpy принимает на вход фрагмент кода функции `lu_vanilla` (листинг Б.1), обрамленный директивами препроцессора `#pragma scop` и `#pragma endscop`.

```
ilpy lu.c > lu.c-deps.txt 2>&1
```

Внешняя переменная `N` трактуется как размер задачи и его вес устанавливается как значение по умолчанию, равное 100. Вычисленные ilpy аффинные отображения имеют вид:

$$\theta_{S_0} = \Big[2k\Big]; \quad \pi_{S_0} = \Big[l-1\Big]; \qquad \theta_{S_1} = \Big[2k+1\Big]; \quad \pi_{S_1} = \Big[i-1\Big].$$

Достаточно одномерного расписания для удовлетворения всех информационных зависимостей. Размещение вычислений обладает свойством вперед направленных коммуникаций.

На листинге Б.2 представлена информация о найденных расписании и размещении вычислений. В круглых скобках указан вес зависимости по данным. `C`

— значение целевой функции (2.5) при найденных $L^{\tau}_{R_i}$ и $L^{\rho}_{R_i}$ для каждого ребра R_i соответственно. Найденное расписание позволяет ограничить $d^{\tau}_{R_i}$ для всех десяти ребер в обобщенном графе зависимостей постоянными величинами, а найденное размещение вычислений позволяет ограничить $d^{\rho}_{R_i}$ нулем только для восьми из десяти ребер в обобщенном графе зависимостей.

На листинге Б.4 представлена программа LU-разложения с синхронным параллелизмом, генерируемая ilpy. Итоговый вариант с постобработкой и расстановкой директив OpenMP представлен на листинге Б.5. Цикл со счетчиком `ilpp` является параллельным — его итерации распределяются между параллельно работающими нитями. Создание нитей происходит единожды перед выполнением вычислений в целях уменьшения накладных расходов на поддержку параллелизма.

Транслятор pluto выдает программный код (листинг Б.9) в соответствии со следующими пространственно-временными отображениями (листинг Б.8):

$$\Phi_{S_0} = \begin{bmatrix} k+l & l & k \end{bmatrix}^T; \quad \Phi_{S_1} = \begin{bmatrix} k+i & i & j \end{bmatrix}^T.$$

Создание и уничтожение нитей внутри цикла по `t1` является неоптимальным, так как влечет накладные расходы на поддержку параллелизма на каждой итерации. В данном случае создание и уничтожение нитей могут быть вынесены за пределы цикла, и с целью обеспечить идентичные условия при сравнении производительности параллельных программ, полученных применением ilpy и pluto, выполняется постобработка результатов работы последнего (листинг Б.10). Конструкция `omp parallel` по возможности выносится из циклов и ветвлений так, чтоб не нарушить корректность программы. Объявления счетчиков циклов, а также границы их изменения, переносятся в заголовки циклов. Последнее служит «косметическим» изменением, приводящим программу к виду, выдаваемому ilpy, в целях сокращения количества строк для упрощения восприятия программистом. Также удаляются объявления неиспользуемых переменных.

На листинге А.22 приведен файл сборки тестового приложения lu для OpenMP. Параллельные варианты программы `lu_vanilla`, полученные применением ilpy (`lu_ilp_sync`) и pluto (`lu_pluto`), запускались с количеством нитей OpenMP, равным 1, 2, 4, 6, 8. Значение параметра `N` задано 3072. Графики ускорения относительно запуска последовательного варианта `lu_vanilla`, а также сравнение двух параллельных вариантов приведены на рисунке 4.1. Подробная статистика запусков приведена в приложении Б.5.

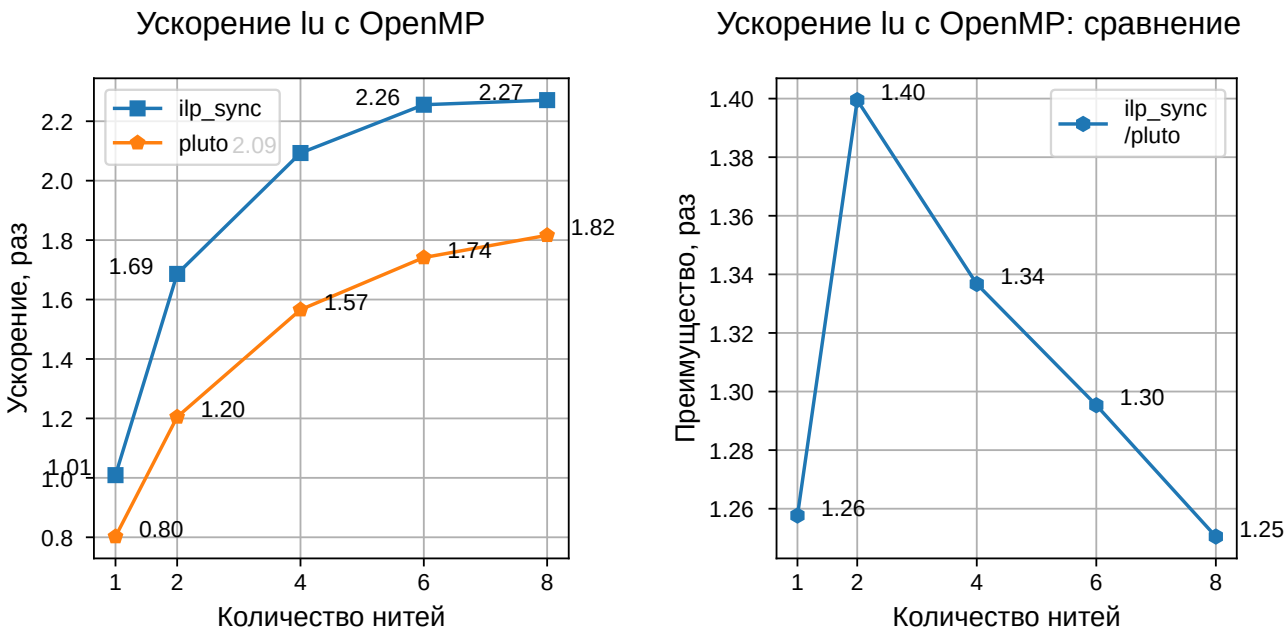


Рисунок 4.1 — Ускорение lu с OpenMP

Параллельный вариант `lu_ilp_sync` показывает лучшую производительность во всех запусках. Наибольшее преимущество перед pluto достигается при запуске в двух нитях и составляет 40%.

4.2.2 Распараллеливание с MPI

Вычисление пространственных и временных отображений осуществляется запуском ilpy с параметром `--arrays`.

```
ilpy lu.c --arrays > lu.c-arrays.txt 2>&1
```

Вычисленные ilpy аффинные отображения имеют вид:

$$\theta_{S_0} = \left[2k\right]; \quad \pi_{S_0} = \left[l\right]; \quad \theta_{S_1} = \left[2k+1\right]; \quad \pi_{S_1} = \left[i\right]; \quad \eta_A = \left[i0\right].$$

В соответствии с ними генерируется параллельная программа, представленная на листинге Б.6. Расписание совпадает с тем, что было найдено для варианта OpenMP. Размещение вычислений отличается только постоянным членом, и также обладает свойством вперед направленных коммуникаций. Найденное размещение массива A можно интерпретировать следующим образом: строка матрицы

A с индексом i располагается на виртуальном процессоре i: $i0$ означает индекс по нулевому измерению массива, что для матрицы указывает на строку.

На листинге Б.3 представлена информация о найденных совместно размещении вычислений и размещении данных. В круглых скобках указан вес доступа к данным. C — значение целевой функции (2.15) при найденных $L^{\rho}_{A_{S_j,i}}$ для каждого доступа к памяти $A_{S_j,i}$ для каждой инструкции S_j. Для пяти из семи доступов к памяти удается ограничить $d^{\rho}_{A_{S_j,i}}$ нулем, что означает отсутствие для данных доступов необходимости организации удаленной записи или удаленного чтения, поскольку совпадает индекс виртуального процессора-вычислителя с индексом виртуального процессора-владельца.

На листинге Б.7 представлена программа LU-разложения с синхронным параллелизмом, дополненная конструкциями для информационного обмена в рамках стандарта MPI. Цикл со счетчиком ilpp является параллельным — его итерации распределяются между параллельно работающими процессами. Реализация операций удаленного чтения выполняется перед параллельным циклом при помощи разработанной библиотеки макросов blockdist.h.

Здесь пересылка одиночного значения A[t0/2][t0/2] выполняется с помощью line_Q_read_vp, при этом задан макрос eta_A_t0_even — размещение элемента массива A[t0/2]. Пересылка подстроки матрицы, определяемая доступом A[(t0-1)/2][k], выполняется применением пары макросов line_Q_read_send и line_Q_read_receive, при этом задан макрос eta_A_t0_odd — размещение элемента массива A[(t0-1)/2]. Вычисления завершаются сбором результата в процессе с рангом 0 с помощью разработанной функции collect_matrix_rows_double.

Транслятор pluto выдает программный код (листинг Б.12) в соответствии со следующими пространственно-временными отображениями (листинг Б.11):

$$\Phi_{S_0} = \begin{bmatrix} k+l & 0 & l & k \end{bmatrix}^T; \quad \Phi_{S_1} = \begin{bmatrix} k+i & 0 & i & j \end{bmatrix}^T.$$

На листинге А.28 приведен файл сборки тестового приложения lu_mpi для MPI. Параллельные варианты программы lu_vanilla, полученные применением ilpy (lu_ilp_arrays) и pluto (lu_pluto_mpi), запускались с количеством процессов MPI, равным 1, 2, 4, 6, 8 в трех вариантах запуска: все процессы на одной машине (суффикс _one), процессы распределены поровну между двумя машинами (суффикс _two), количество машин равно количеству процессов (суффикс

_eq). В каждом процессе MPI поддерживалась единая рабочая нить. Значение параметра `N` задано 3072. Графики ускорения относительно запуска последовательного варианта `lu_vanilla`, а также сравнение двух параллельных вариантов приведены на рисунке 4.2.

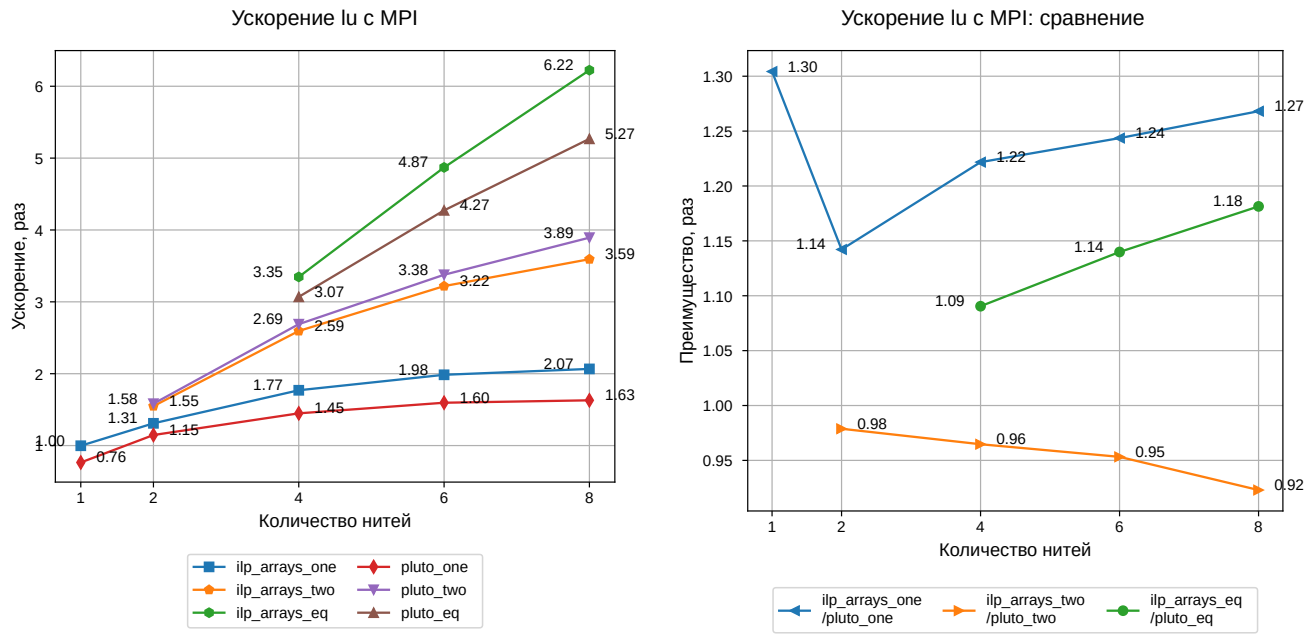


Рисунок 4.2 — Ускорение lu с MPI

Все решения позволили получить ускорение вычислений. Параллельный вариант `lu_ilp_arrays` показывает лучшую производительность во всех запусках, кроме выполняемых на двух машинах. Проигрыш в производительности составил от 2% до 8% в зависимости от количества процессов. Наибольшее преимущество перед pluto достигается при запуске в восьми процессах и составляет 27% для запуска на одной машине и 18% для запуска на восьми машинах. Подробная статистика запусков приведена в приложении Б.6.

На рисунке 4.3 представлены графики, иллюстрирующие долю времени вычислений во всех запусках, а также приводится сравнение соответственных вариантов, полученных применением ilpy и pluto.

Решения pluto во всех параллельных запусках демонстрируют лучшую загруженность процессоров, однако преимущество перед ilpy достигается только при запуске на двух машинах. Сильная сторона pluto — возможность динамического распределения нагрузки между процессорами в функции `polyrt_loop_dist`. На рисунке 4.4 проиллюстрировано распределение времени

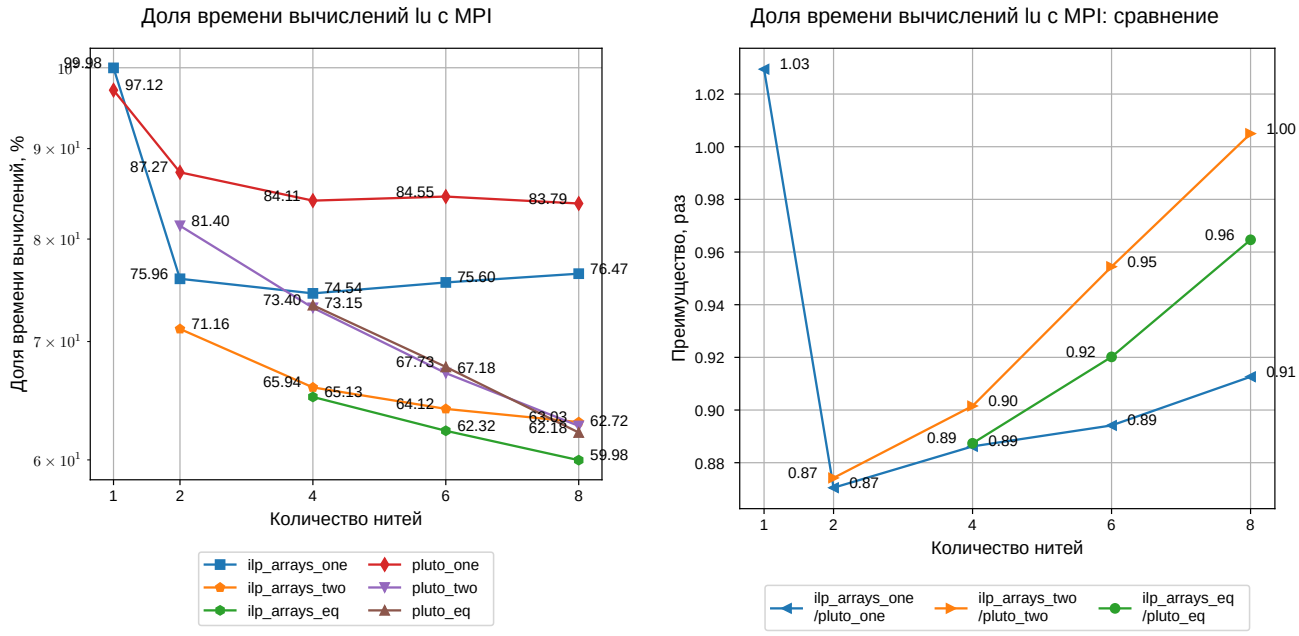


Рисунок 4.3 — Доля времени вычислений в запусках lu с MPI

исполнения кода между вычислениями, информационным обменом и барьерной синхронизацией.

Во всех запусках решения pluto выглядят более сбалансированными, в то время как решения ilpy демонстрируют меньшую загруженность вычислениями и более весомые накладные расходы в процессах с меньшим рангом, и большую загруженность вычислениями и менее весомые накладные расходы в процессах с большим рангом. Это справедливо при уменьшении количества процессов с восьми до двух.

4.3 Матричное произведение atax

4.3.1 Распараллеливание с OpenMP

Рассматривается программа из набора тестов polybench [110] (листинг В.1). В дальнейшем данный вариант будет называться atax_p. Также рассматривается модифицированный, но сохраняющий корректность вычислений вариант программы (листинг В.2). Он требует меньше памяти для выполнения, но уби-

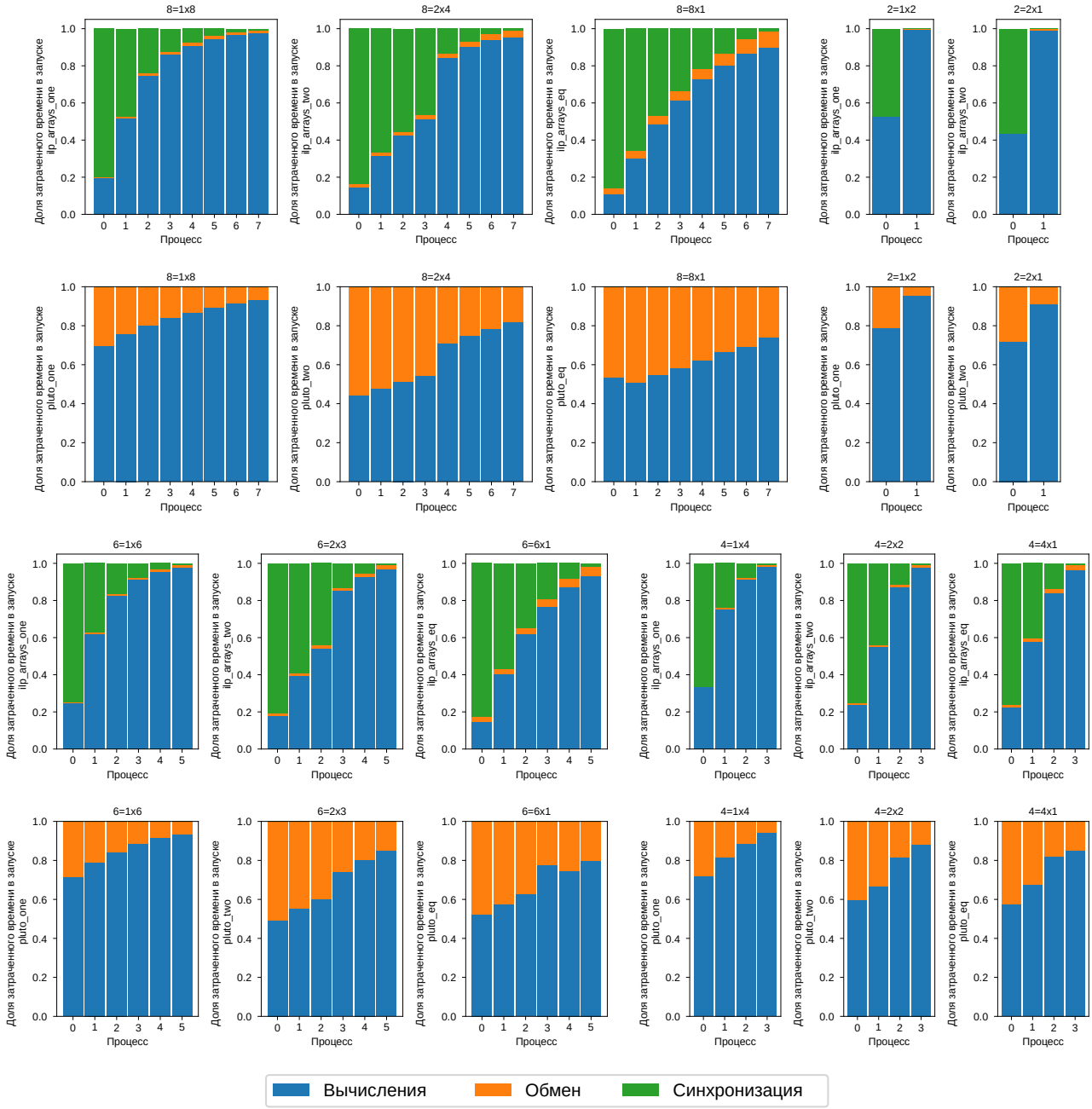


Рисунок 4.4 — Разнородная нагрузка при запуске параллельных вариантов lu с MPI

рает возможность параллельного вычисления элементов массива `tmp`: массив заменяется на переменную. Домены инструкций не претерпели изменений. В дальнейшем данный вариант будет называться atax. Разработанный транслятор ilpy принимает на вход тексты `atax_p` и `atax`.

```
ilpy atax_p.c --cloog-f 1 > atax_p.c-deps.txt 2>&1
ilpy atax.c --cloog-f 1 --ilp-col-ub 2 > atax.c-deps.txt 2>&1
```

Параметр cloog «начальная глубина цикла для оптимизации потока управления» устанавливается в умалчиваемое значение 1. Для варианта `atax` в целях

ускорения решения задачи ЛЦП при вычислении размещения вычислений устанавливается верхняя граница для каждой переменной, равная 2.

Внешние переменные `M` и `N` трактуются как размер задачи и их вес устанавливается как значение по умолчанию, равное 100.

Вычисленные ilpy для `atax_p` аффинные отображения имеют вид:

$$\theta_{S_0} = \left[N\right]; \quad \pi_{S_0} = \left[-i+N-1\right]; \qquad \theta_{S_1} = \left[i\right]; \quad \pi_{S_1} = \left[i\right];$$
$$\theta_{S_2} = \left[i+j+1\right]; \quad \pi_{S_2} = \left[i\right]; \qquad \theta_{S_3} = \left[i+N+1\right]; \quad \pi_{S_3} = \left[i-j+N\right].$$

Достаточно одномерного расписания для удовлетворения всех информационных зависимостей в `atax_p`, и размещение вычислений обладает свойством вперед направленных коммуникаций. На листинге В.3 представлена информация о найденных расписании и размещении вычислений соответственно. Найденное расписание позволяет ограничить $d^{\tau}_{R_i}$ для шести из двенадцати ребер в обобщенном графе зависимостей постоянными величинами. Найденное размещение вычислений позволяет ограничить $d^{\rho}_{R_i}$ постоянной величиной для восьми из двенадцати ребер в обобщенном графе зависимостей.

Параллельный вариант `atax_p`, сгенерированный ilpy, представлен на листинге В.8. Итоговый вариант с постобработкой и директивами OpenMP представлен на листинге В.9. Транслятор pluto выдает программный код (листинг В.20) в соответствии со следующими пространственно-временными отображениями (листинг В.19):

$$\Phi_{S_0} = \begin{bmatrix} 2 \\ i \\ 0 \end{bmatrix}; \quad \Phi_{S_1} = \begin{bmatrix} 0 \\ i \\ 0 \end{bmatrix}; \quad \Phi_{S_2} = \begin{bmatrix} 1 \\ i \\ j \end{bmatrix}; \quad \Phi_{S_3} = \begin{bmatrix} 3 \\ i+j \\ j \end{bmatrix}.$$

Вариант с постобработкой результатов работы pluto представлен на листинге В.21).

Вычисленные ilpy для `atax` аффинные отображения имеют вид:

$$\theta_{S_0} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}; \quad \pi_{S_0} = \left[-i+N-1\right]; \qquad \theta_{S_1} = \begin{bmatrix} 3i+1 \\ 0 \end{bmatrix}; \quad \pi_{S_1} = \left[i\right];$$
$$\theta_{S_2} = \begin{bmatrix} 3i+2 \\ j \end{bmatrix}; \quad \pi_{S_2} = \left[i\right]; \qquad \theta_{S_3} = \begin{bmatrix} 3i+3 \\ 0 \end{bmatrix}; \quad \pi_{S_3} = \left[i-j+N\right].$$

Потребовалось двумерное расписание для `atax`, и размещение вычислений не обладает свойством вперед направленных коммуникаций. На листинге В.4

представлена информация о найденных расписании и размещении вычислений соответственно. Найденное расписание обладает следующими свойствами: первый компонент удовлетворяет семнадцать зависимостей из двадцати и для пятнадцати из них позволяет ограничить $d^{\tau}_{R_i}$ постоянной величиной; второй компонент удовлетворяет оставшиеся три зависимости, и для каждой из них позволяет ограничить $d^{\tau}_{R_i}$ постоянной величиной. Найденное размещение вычислений позволяет ограничить $d^{\rho}_{R_i}$ постоянной величиной для четырнадцати из двадцати ребер в обобщенном графе зависимостей.

Параллельный вариант `atax`, сгенерированный ilpy, представлен на листинге В.10. Итоговый вариант с постобработкой и директивами OpenMP представлен на листинге В.11. Транслятор pluto выдает программный код (листинг В.23) в соответствии со следующими пространственно-временными отображениями (листинг В.22):

$$\Phi_{S_0} = \begin{bmatrix} 0 \\ i \\ 2 \\ 1 \\ 0 \end{bmatrix}; \quad \Phi_{S_1} = \begin{bmatrix} 1 \\ i \\ 0 \\ 1 \\ 0 \end{bmatrix}; \quad \Phi_{S_2} = \begin{bmatrix} 1 \\ i \\ 1 \\ 0 \\ j \end{bmatrix}; \quad \Phi_{S_3} = \begin{bmatrix} 1 \\ i \\ 1 \\ 1 \\ j \end{bmatrix}.$$

Вариант с постобработкой результатов работы pluto представлен на листинге В.24).

Рассмотрим еще один вариант распараллеливания программы `atax_p`:

```
ilpy atax_p.c --smdp --out-fn-suffix deps_mdp > atax_p.c-deps_mdp.txt 2>&1
```

С параметром `--smdp` предполагается отыскание параллельной формы на основе топологической сортировки вершин обобщенного графа зависимостей:

$$\theta_{S_0} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}; \quad \pi_{S_0} = \begin{bmatrix} -i + N - 1 \end{bmatrix}; \qquad \theta_{S_1} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}; \quad \pi_{S_1} = \begin{bmatrix} i \end{bmatrix};$$

$$\theta_{S_2} = \begin{bmatrix} 1 \\ j \end{bmatrix}; \quad \pi_{S_2} = \begin{bmatrix} i \end{bmatrix}; \qquad \theta_{S_3} = \begin{bmatrix} 2 \\ i \end{bmatrix}; \quad \pi_{S_3} = \begin{bmatrix} i - j + N \end{bmatrix}.$$

Первый компонент многомерного расписания получен на основе топологической сортировки инструкций. Второго компонента расписания оказалось достаточно для удовлетворения всех оставшихся информационных зависимостей (представляющих петли в обобщенном графе зависимостей) в `atax_p`. Размещение вычислений обладает свойством вперед направленных коммуникаций. На

листинге В.5 представлена информация о найденных расписании и размещении вычислений соответственно. Второй компонент расписания позволяет ограничить $d^{\tau}_{R_i}$ для всех оставшихся шести ребер в обобщенном графе зависимостей постоянными величинами. Найденное размещение вычислений позволяет ограничить $d^{\rho}_{R_i}$ постоянной величиной для восьми из двенадцати ребер в обобщенном графе зависимостей.

Параллельный вариант `atax_p`, сгенерированный ilpy, представлен на листинге В.12. Итоговый вариант с постобработкой и директивами OpenMP представлен на листинге В.13.

На листинге А.23 приведен файл сборки тестового приложения atax для OpenMP. Параллельные варианты программы atax, полученные применением ilpy (`atax_p_ilp_sync`, `atax_ilp_sync`, `atax_p_ilp_sync_mdp`) и pluto (`atax_p_pluto`, `atax_pluto`), запускались с количеством нитей OpenMP, равным 1, 2, 4, 6, 8. Значения параметров: `N` задано 6144, `M` задано 8192. Графики ускорения относительно запуска последовательного варианта `atax_p`, а также сравнение параллельных вариантов приведены на рисунке 4.5. Подробная статистика запусков приведена в приложении В.5.

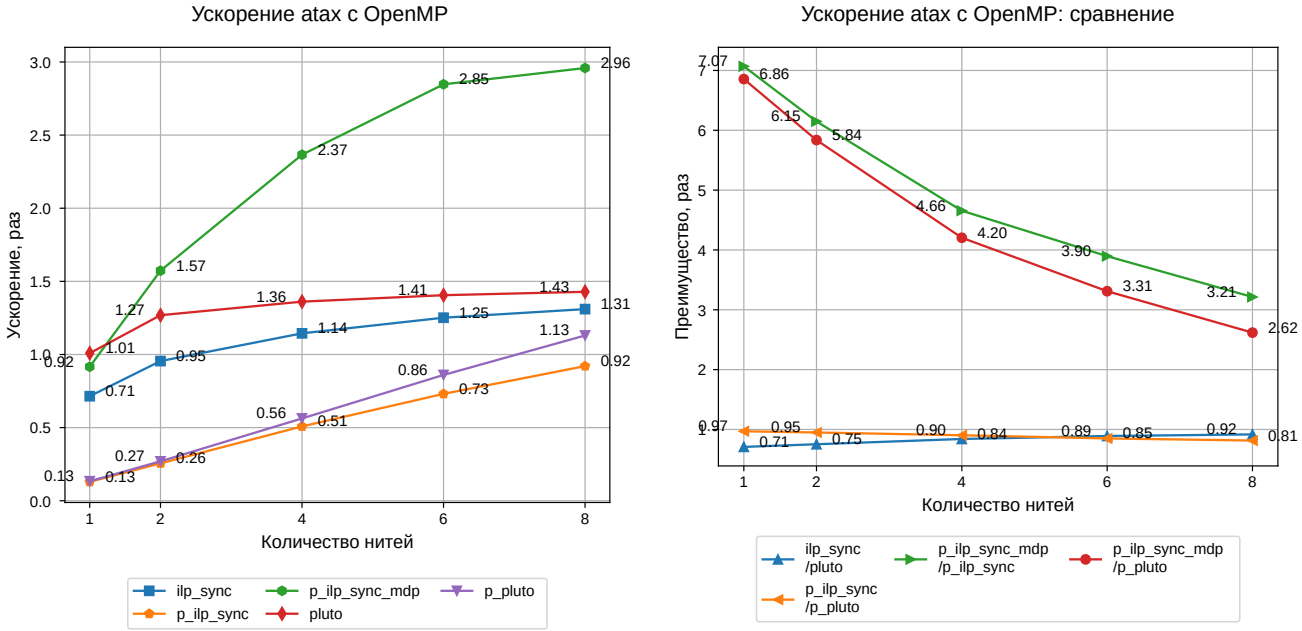


Рисунок 4.5 — Ускорение atax с OpenMP

Параллельный вариант `atax_p_ilp_sync_mdp` показывает лучшую производительность во всех запусках. Рассмотренный пример позволяет заключить, что стремление улучшить локальность использования данных, сближая операции

во времени и пространстве виртуальных процессоров, не обязательно приведет к наилучшей производительности, так как паттерны доступа к массивам могут оказаться неудачными для применяемой процессорной микроархитектуры.

4.3.2 Распараллеливание с MPI

Вычисление пространственных и временных отображений осуществляется запуском ilpy с параметром `--arrays`.

```
ilpy atax_p.c --arrays --cloog-f 1 > atax_p.c-arrays.txt 2>&1
```

Вычисленные ilpy аффинные отображения имеют вид:

$$\theta_{S_0} = \left[N\right]; \quad \pi_{S_0} = \left[i\right]; \quad \theta_{S_1} = \left[i\right]; \quad \pi_{S_1} = \left[i\right];$$
$$\theta_{S_2} = \left[i+j+1\right]; \quad \pi_{S_2} = \left[j\right]; \quad \theta_{S_3} = \left[i+N+1\right]; \quad \pi_{S_3} = \left[j\right];$$
$$\eta_y = \left[i0\right]; \quad \eta_A = \left[i1\right]; \quad \eta_{tmp} = \left[0\right]; \quad \eta_x = \left[i0\right].$$

В соответствии с ними генерируется параллельная программа, представленная на листинге В.14. Итоговый вариант с постобработкой и вызовами MPI представлен на листинге В.15. Макросы распределения данных представлены на листинге В.18. Реализация операций удаленного доступа выполняется при помощи разработанной библиотеки макросов blockdist.h. Вычисления завершаются сбором результата в процессе с рангом 0 с помощью разработанной функции `collect_matrix_cols_double`. Расписание совпадает с тем, что было найдено для варианта OpenMP. Размещение вычислений не обладает свойством вперед направленных коммуникаций. Найденное размещение массива `A` можно интерпретировать следующим образом: столбец матрицы `A` с индексом `i` располагается на виртуальном процессоре `i`: $i1$ означает индекс по первому измерению массива, что для матрицы указывает на столбец. Массив `tmp` полностью размещен на виртуальном процессоре с индексом 0. Массивы `x` и `y` распределены между виртуальными процессорами поэлементно: элемент с индексом `i` располагается на виртуальном процессоре `i`. На листинге В.6 представлена информация о найденных совместно размещении вычислений и размещении данных: для шести из

десяти доступов к памяти удается ограничить $d^{\rho}_{A_{S_j,i}}$ нулем, что означает отсутствие для данных доступов необходимости организации удаленной записи или удаленного чтения.

Транслятор pluto выдает программный код (листинг В.26) в соответствии со следующими пространственно-временными отображениями (листинг В.25):

$$\Phi_{S_0} = \begin{bmatrix} 2 & 0 & 0 & 0 & i & 0 & 0 \end{bmatrix}^T; \qquad \Phi_{S_1} = \begin{bmatrix} 0 & 0 & 0 & 0 & i & 0 & 0 \end{bmatrix}^T;$$

$$\Phi_{S_2} = \begin{bmatrix} 1 & 0 & 0 & 0 & i & 0 & j \end{bmatrix}^T; \qquad \Phi_{S_3} = \begin{bmatrix} 3 & 0 & 0 & 0 & i+j & 0 & j \end{bmatrix}^T.$$

На листинге А.29 приведен файл сборки тестового приложения atax_p_mpi для MPI. Параллельные варианты программы `atax_p`, полученные применением ilpy (`atax_p_ilp_arrays`) и pluto (`atax_p_pluto_mpi`), запускались с количеством процессов MPI, равным 1, 2, 4, 6, 8 в трех вариантах запуска: все процессы на одной машине (суффикс `_one`), процессы распределены поровну между двумя машинами (суффикс `_two`), количество машин равно количеству процессов (суффикс `_eq`). В каждом процессе MPI поддерживалась единая рабочая нить. Значения параметров: `N` задано 6144, `M` задано 8192. Графики ускорения относительно запуска последовательного варианта `atax_p`, а также сравнение параллельных вариантов приведены на рисунке 4.6.

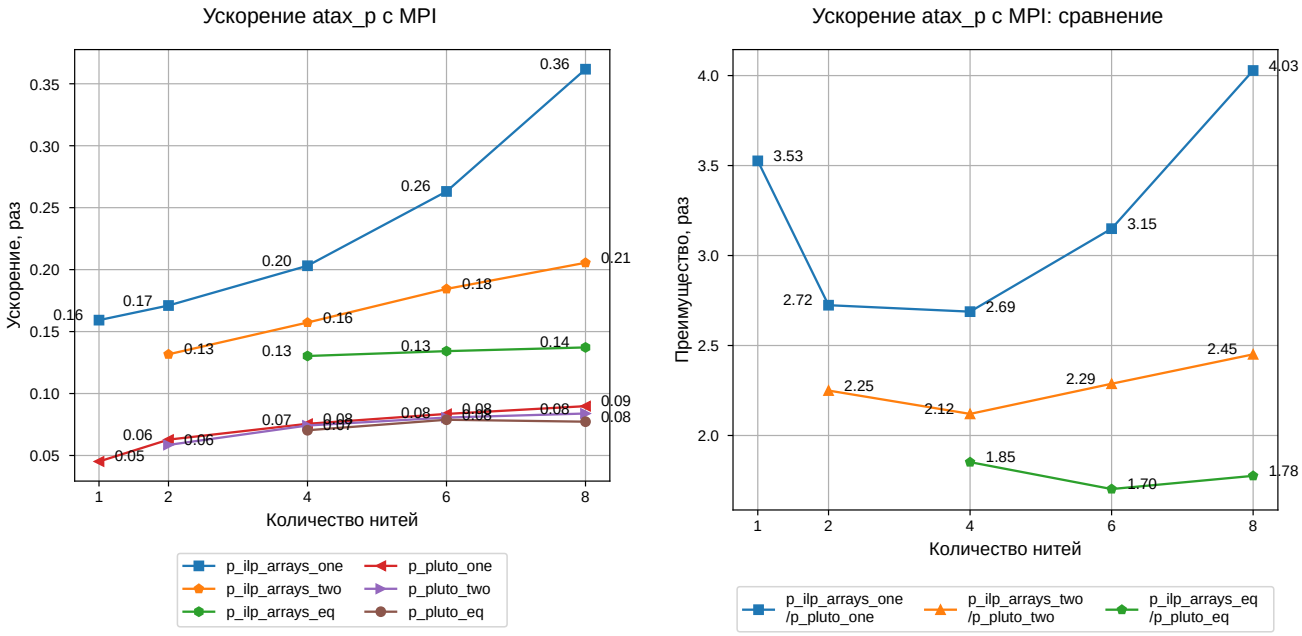


Рисунок 4.6 — Ускорение atax с MPI (`ilpy --arrays`)

Никакие решения не позволили получить ускорение вычислений. Параллельный вариант `atax_p_ilp_arrays` показывает лучшую производительность

во всех запусках. Наибольшее преимущество перед pluto достигается при запуске в восьми процессах и составляет 4 раза для запуска на одной машине и 2,45 раза для запуска на двух машинах. Подробная статистика запусков приведена в приложении В.6.

На рисунке 4.7 представлены графики, иллюстрирующие долю времени вычислений во всех запусках, а также приводится сравнение соответственных вариантов, полученных применением ilpy и pluto. Решения pluto во всех параллельных запусках демонстрируют худшую загруженность процессоров.

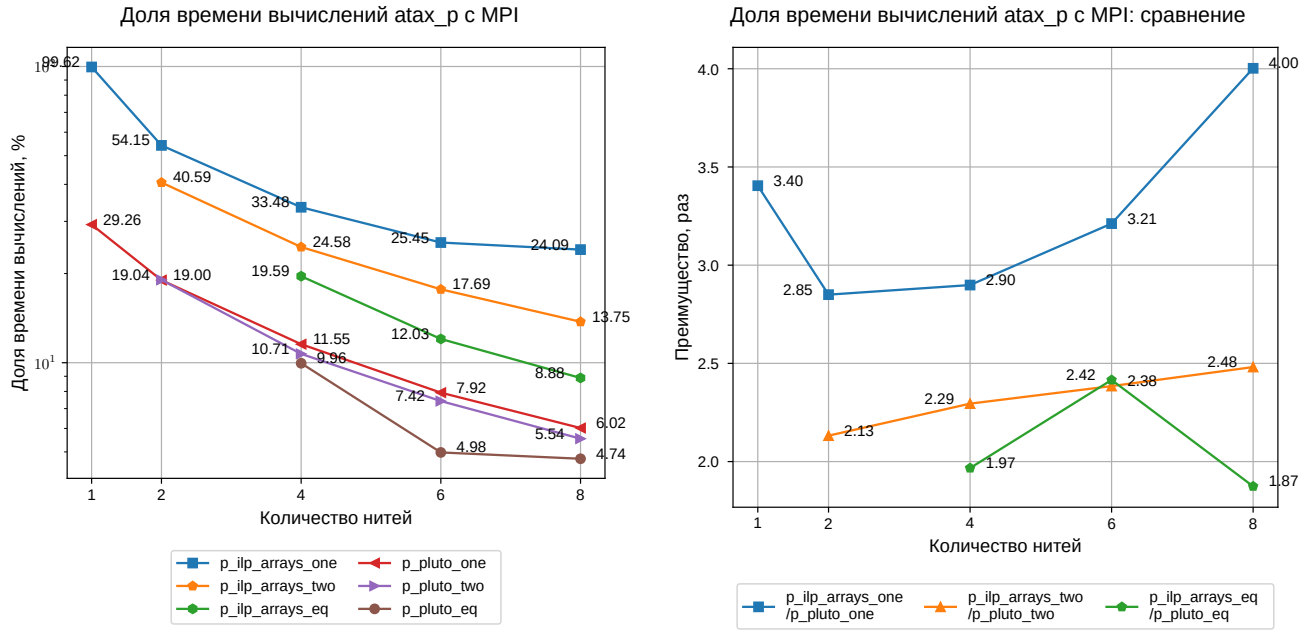


Рисунок 4.7 — Доля времени вычислений в запусках atax с MPI (`ilpy --arrays`)

На рисунке 4.8 проиллюстрировано распределение времени исполнения кода между вычислениями, информационным обменом и барьерной синхронизацией. Во всех запусках решения pluto выглядят более сбалансированными, но большая доля времени исполнения затрачена на весомые накладные расходы. Это справедливо при уменьшении количества процессов с восьми до двух.

Также рассмотрим вариант с параллельной формой, полученной применением топологической сортировки вершин обобщенного графа зависимостей:

```
ilpy atax_p.c --arrays --smdp --out-fn-suffix arrays_mdp > atax_p.c-
   arrays_mdp.txt 2>&1
```

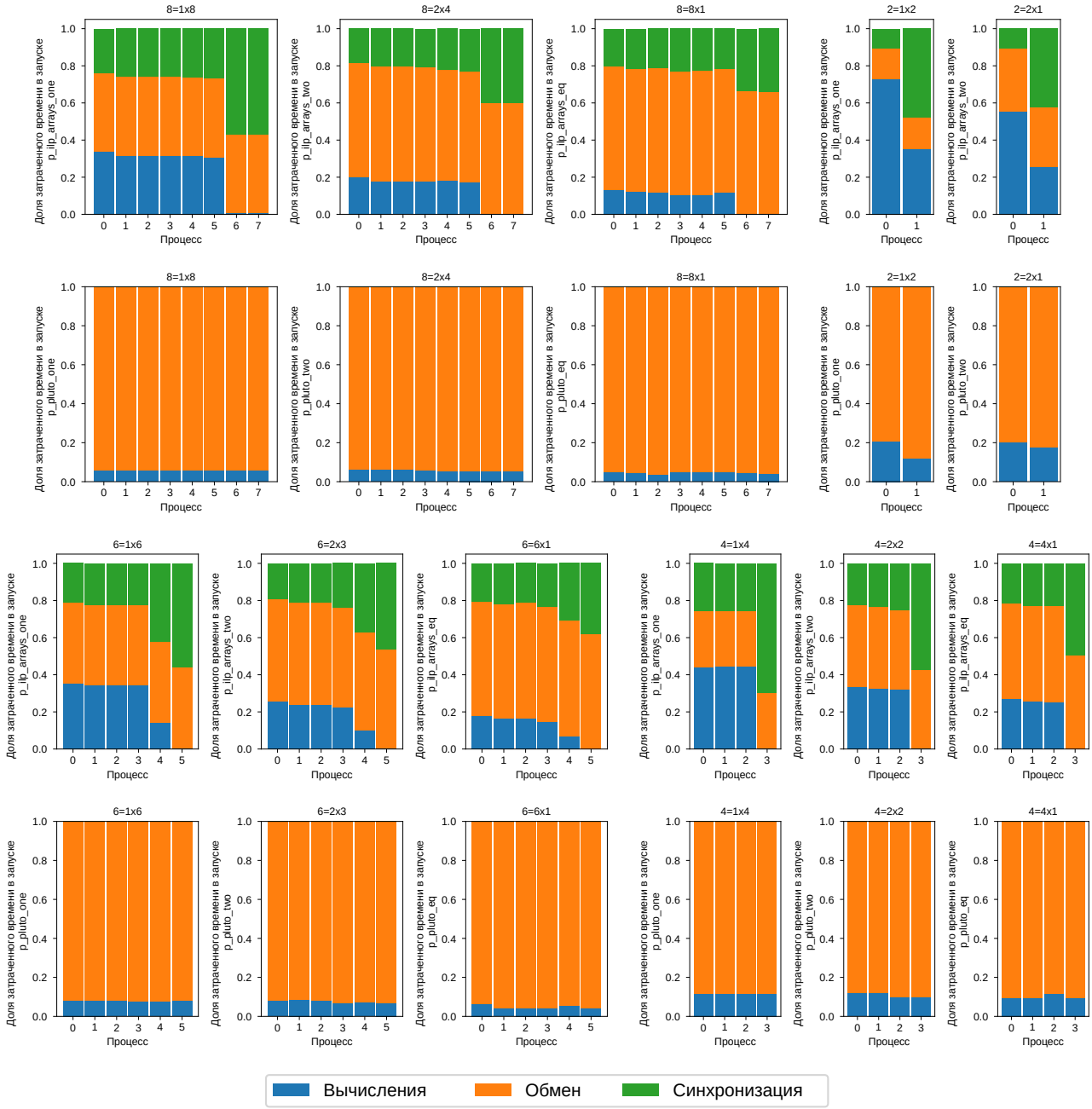


Рисунок 4.8 — Разнородная нагрузка при запуске параллельных вариантов atax с MPI (`ilpy --arrays`)

Вычисленные ilpy аффинные отображения имеют вид:

$$\theta_{S_0} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}; \quad \pi_{S_0} = \begin{bmatrix} -i + N - 1 \end{bmatrix}; \qquad \theta_{S_1} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}; \quad \pi_{S_1} = \begin{bmatrix} -i + M - 1 \end{bmatrix};$$

$$\theta_{S_2} = \begin{bmatrix} 1 \\ j \end{bmatrix}; \quad \pi_{S_2} = \begin{bmatrix} -i + M - 1 \end{bmatrix}; \qquad \theta_{S_3} = \begin{bmatrix} 2 \\ i \end{bmatrix}; \quad \pi_{S_3} = \begin{bmatrix} -j + N - 1 \end{bmatrix};$$

$$\eta_y = \begin{bmatrix} -i0 + N - 1 \end{bmatrix}; \quad \eta_A = \begin{bmatrix} -i1 + N - 1 \end{bmatrix};$$

$$\eta_{tmp} = \begin{bmatrix} -i0 + M - 1 \end{bmatrix}; \quad \eta_x = \begin{bmatrix} M - 1 \end{bmatrix}.$$

В соответствии с ними генерируется параллельная программа, представленная на листинге В.16. Итоговый вариант с постобработкой и вызовами MPI представлен на листинге В.17. Макросы распределения данных представлены на листинге В.18. Введены дополнительные флаги, изменяющие работу программы в аспектах информационного обмена:

– `use_bcast`: использовать широковещательную передачу (значение True, суффикс _bcast) или двухстороннюю коммуникацию процессов из blockdist.h (значение False, суффикс _sendrecv);
– `dist_input`: входные данные распределены в соответствии с размещением данных (значение True, суффикс _distinp) или раскопированы на всех процессах (значение False, суффикс _dupinp).

Вычисления завершаются сбором результата в процессе с рангом 0 с помощью разработанной функции `collect_matrix_cols_double`. Расписание совпадает с тем, что было найдено для варианта OpenMP. Размещение вычислений не обладает свойством вперед направленных коммуникаций. Найденное размещение данных можно интерпретировать следующим образом:

– столбец матрицы `A` с индексом `i` располагается на виртуальном процессоре `-i+N-1`;
– массив `x` полностью размещен на виртуальном процессоре с индексом `M-1`;
– элемент массива `y` с индексом `i` располагается на виртуальном процессоре `-i+N-1`;
– элемент массива `tmp` с индексом `i` располагается на виртуальном процессоре `-i+M-1`.

На листинге В.7 представлена информация о найденных совместно размещении вычислений и размещении данных: для семи из десяти доступов к памяти удается ограничить $d^{\rho}_{A_{S_j,i}}$ нулем, что означает отсутствие для данных доступов необходимости организации удаленной записи или удаленного чтения.

Графики ускорения относительно запуска последовательного варианта `atax_p`, а также сравнение параллельных вариантов приведены на рисунках 4.9 и 4.10.

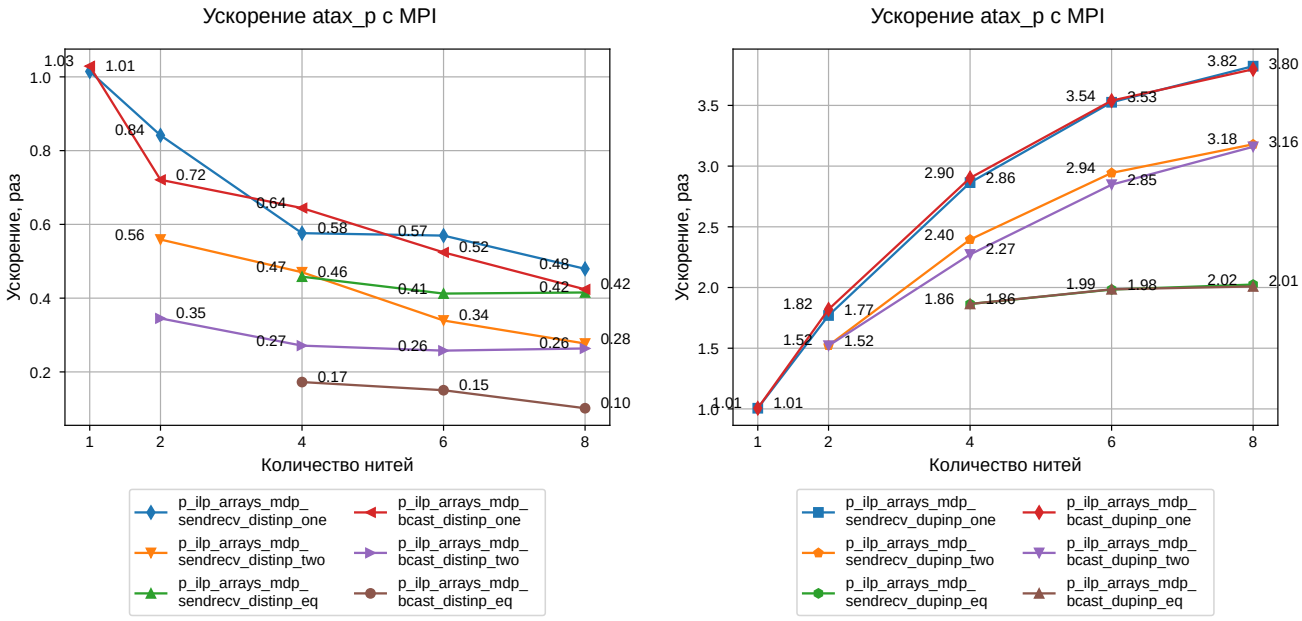


Рисунок 4.9 — Ускорение atax с MPI (`ilpy --arrays --smdp`)

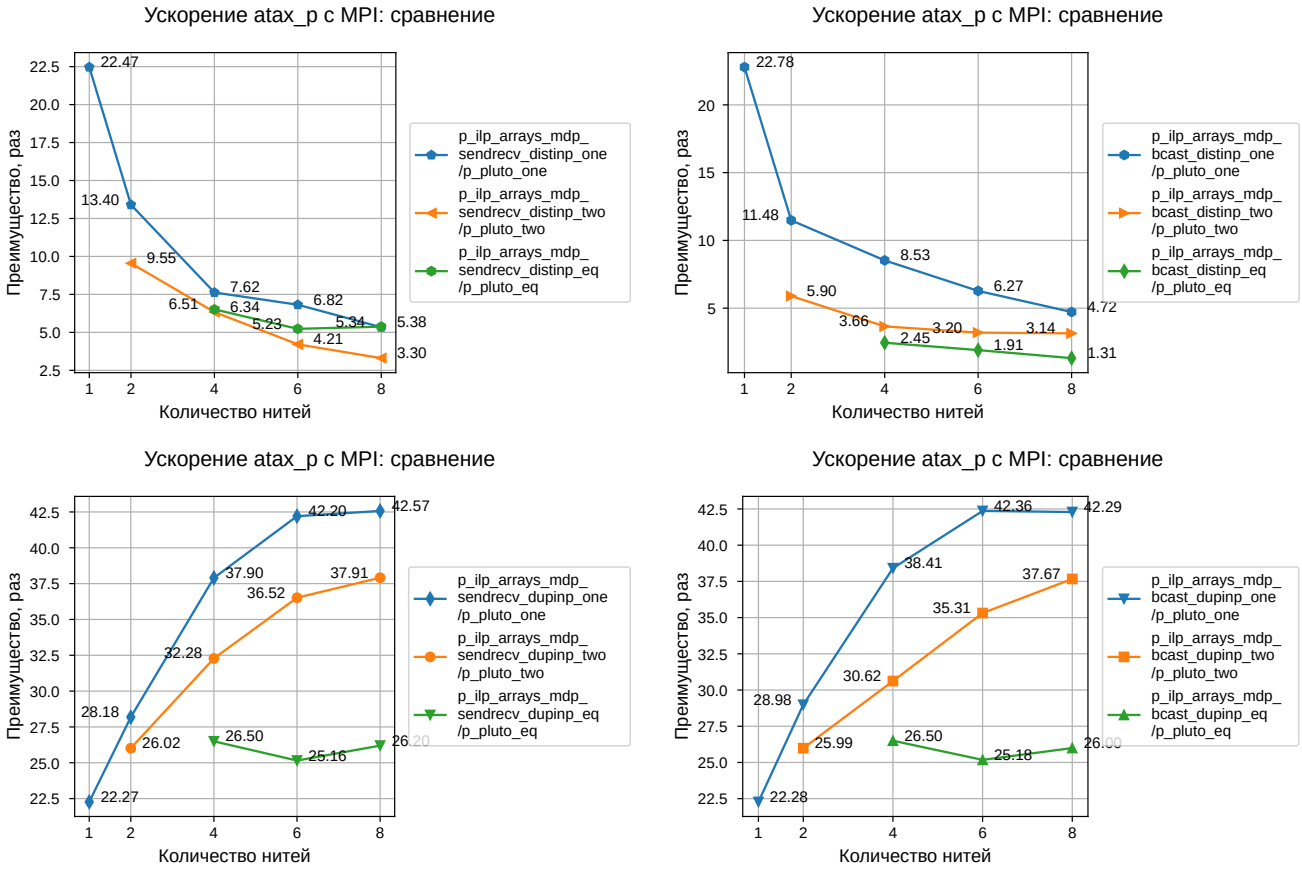


Рисунок 4.10 — Сравнение производительности параллельных вариантов atax с MPI (`ilpy --arrays --smdp`)

Никакие решения с распределенными входными данными не позволили получить ускорение вычислений, но все решения с дублированными входными данными позволили это сделать. Применение широковещательной пересылки данных заметно повлияло на быстродействие только при распределенных входных данных: при запуске 4 процессов на одной машине наблюдается выигрыш в быстродействии по сравнению с макросами blockdist.h, но во всех остальных запусках быстродействие ухудшилось. Подробная статистика запусков приведена в приложении В.6.

На рисунке 4.11 представлены графики, иллюстрирующие долю времени вычислений во всех запусках с дублированными входными данными, а также приводится сравнение соответственных вариантов, полученных применением ilpy (`dist_input == False`, `use_bcast == False`) и pluto. Решения pluto во всех параллельных запусках демонстрируют худшую загруженность процессоров.

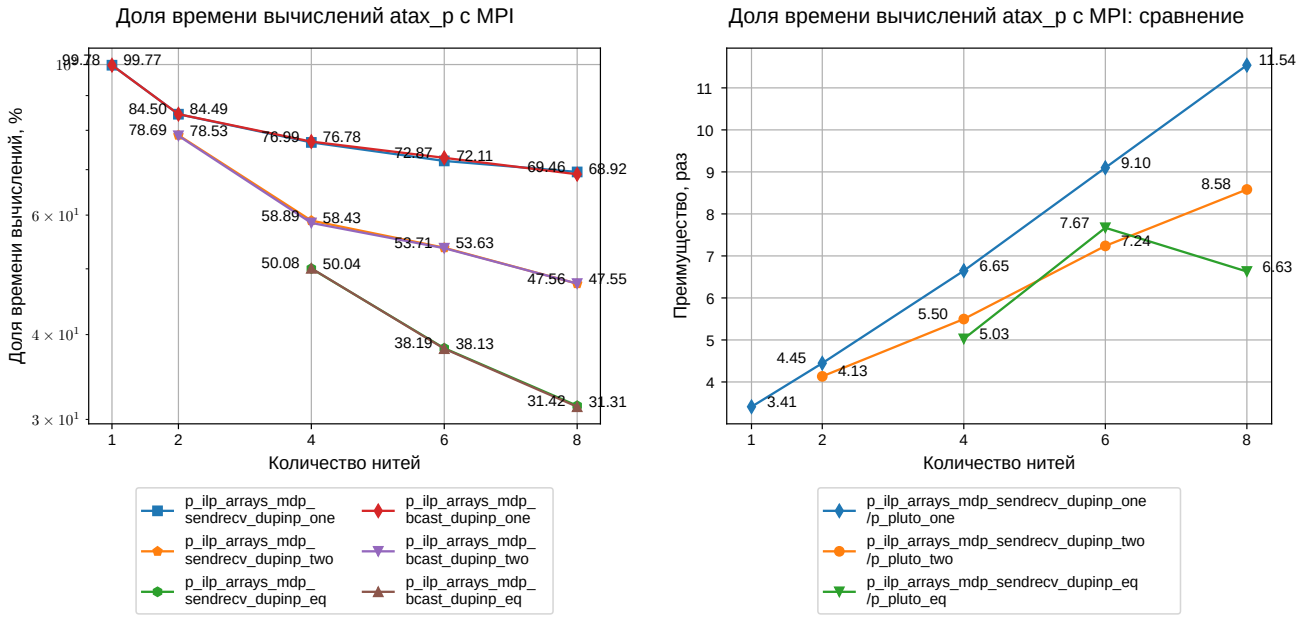


Рисунок 4.11 — Доля времени вычислений в запусках atax с MPI (`ilpy --arrays --smdp`)

На рисунке 4.12 проиллюстрировано распределение времени исполнения кода между вычислениями, информационным обменом и барьерной синхронизацией. Во всех запусках решения pluto выглядят более сбалансированными, но большая доля времени исполнения затрачена на весомые накладные расходы. Это справедливо при уменьшении количества процессов с восьми до двух.

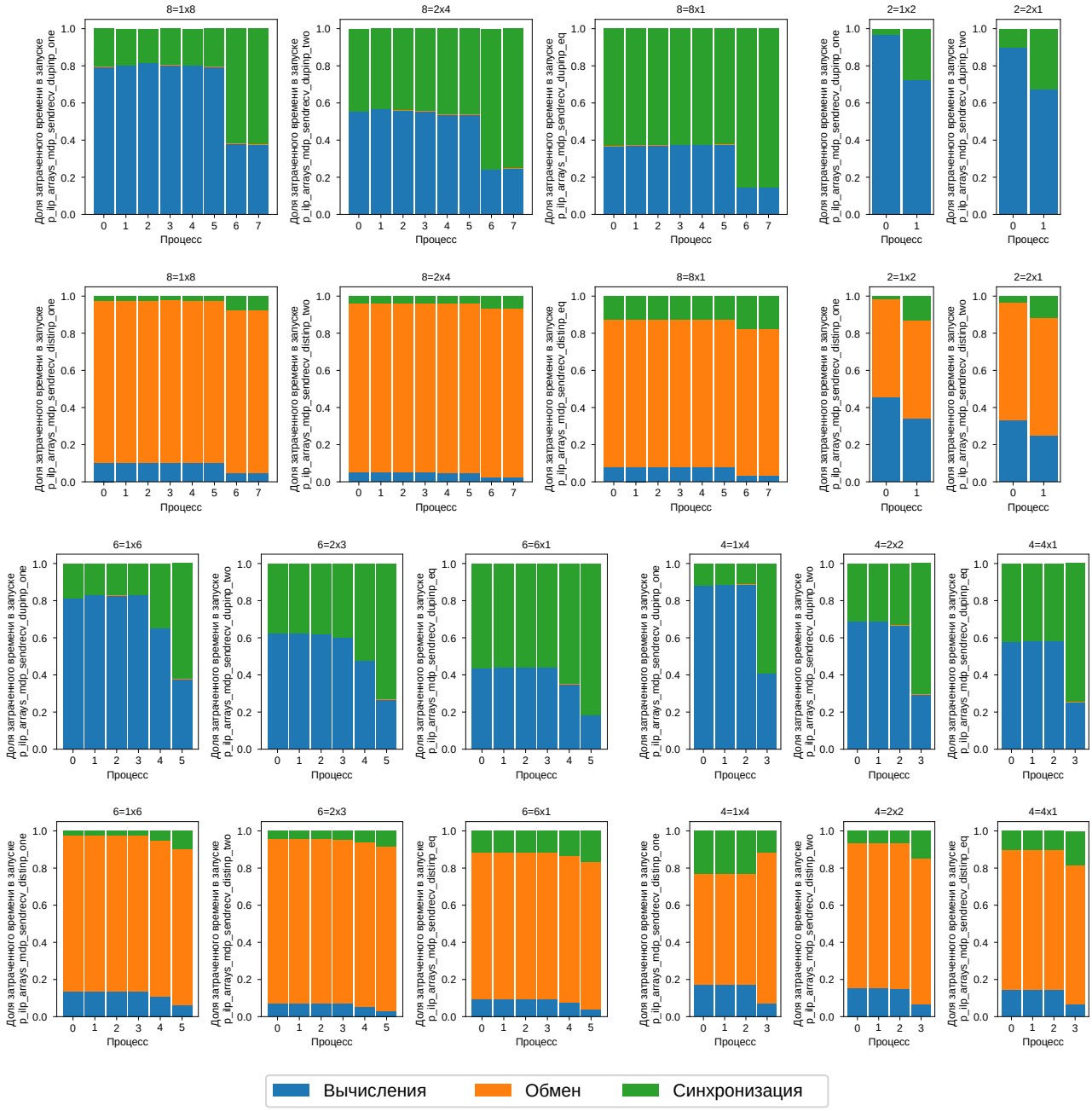


Рисунок 4.12 — Разнородная нагрузка при запуске параллельных вариантов atax с MPI (`--arrays --smdp`)

Рассмотренный пример показывает, что дублирование входных данных во всех вычислительных процессах может повысить производительность результирующей параллельной программы за счет снижения интенсивности информационного обмена, если достаточно оперативной памяти для размещения массивов. Такая оптимизация допустима только для массивов, в которые не производится запись.

4.4 Процедура syr2k

4.4.1 Распараллеливание с OpenMP

Рассматривается программа из набора тестов polybench [110], выполняющая вычисления «на месте», то есть результат окажется в той же области памяти, что и входные данные (функция `syr2k_vanilla`, листинг Г.1). Разработанный транслятор ilpy принимает на вход фрагмент кода функции `syr2k_vanilla` (листинг Г.1), обрамленный директивами препроцессора `#pragma scop` и `#pragma endscop` в упрощенном виде: исключаются скаляры `alpha` и `beta` для упрощения анализа программы. На этапах постобработки они будут возвращены простой текстовой заменой.

```
ilpy syr2k.c > syr2k.c-deps.txt 2>&1
```

Внешние переменные `N` и `M` трактуются как размер задачи и их вес устанавливается как значение по умолчанию, равное 100. Вычисленные ilpy аффинные отображения имеют вид:

$$\theta_{S_0} = \left[0\right]; \qquad \pi_{S_0} = \left[i\right];$$
$$\theta_{S_1} = \left[k+1\right]; \qquad \pi_{S_1} = \left[i\right].$$

Достаточно одномерного расписания для удовлетворения всех информационных зависимостей. Размещение вычислений обладает свойством вперед направленных коммуникаций. На листинге Г.2 представлена информация о найденных расписании и размещении вычислений соответственно. Найденное расписание позволяет ограничить $d^{\tau}_{R_i}$ для трех из семи ребер в обобщенном графе зависимостей постоянными величинами, а найденное размещение вычислений позволяет ограничить $d^{\rho}_{R_i}$ нулем для всех семи ребер в обобщенном графе зависимостей.

На листинге 4.1 представлена программа syr2k с синхронным параллелизмом, генерируемая ilpy.

Можно заметить, что локальность использования данных улучшится, если цикл по `t0` вложить в цикл по `k` — корректность программы не нарушится, и получится программа с асинхронным параллелизмом. ilpy поддерживает генерацию таких программ с параметром `--async`.

```
ilpy syr2k.c --async > syr2k.c-adeps.txt 2>&1
```

Листинг 4.1 Процедура syr2k (параллельный вариант ilpy, синхронный параллелизм) на языке C

```
if (N >= 1) {
 for (ilpp=0;ilpp<=N-1;ilpp++) {
  for (j=0;j<=ilpp;j++) {
   C[ilpp][j] += C[ilpp][j];
  }
 }
 for (t0=1;t0<=M;t0++) {
  for (ilpp=0;ilpp<=N-1;ilpp++) {
   for (k=0;k<=ilpp;k++) {
    C[ilpp][k] += A[k][t0-1]*B[ilpp][t0-1] + B[k][t0-1]*A[ilpp][t0-1];
   }
  }
 }
}
```

Вычисленные ilpy аффинные отображения имеют вид:

$$\theta_{S_0} = \begin{bmatrix} 0 \end{bmatrix}; \qquad \pi_{S_0} = \begin{bmatrix} i \\ -4i + j + 12N - 12 \end{bmatrix};$$

$$\theta_{S_1} = \begin{bmatrix} k + 1 \end{bmatrix}; \qquad \pi_{S_1} = \begin{bmatrix} i \\ -4i + j + 12N - 12 \end{bmatrix}.$$

Размещение вычислений теперь имеет два измерения. Свойство вперед направленных коммуникаций не нарушено. На листинге Г.3 представлена информация о найденном размещении вычислений. Каждый компонент найденного размещения вычислений позволяет ограничить $d^{\rho}_{R_i}$ нулем для всех семи ребер в обобщенном графе зависимостей.

На листинге Г.5 представлена программа syr2k с асинхронным параллелизмом, генерируемая ilpy. Итоговый вариант с постобработкой и расстановкой директив OpenMP представлен на листинге Г.6.

Транслятор pluto выдает программный код (листинг Г.10) в соответствии со следующими пространственно-временными отображениями (листинг Г.9):

$$\Phi_{S_0} = \begin{bmatrix} 0 \\ i \\ j \\ 0 \end{bmatrix}; \quad \Phi_{S_1} = \begin{bmatrix} 1 \\ i \\ j \\ k \end{bmatrix}.$$

Вариант с постобработкой результатов работы pluto представлен на листинге Г.11).

На листинге А.24 приведен файл сборки тестового приложения syr2k для OpenMP. Параллельные варианты программы `syr2k_vanilla`, полученные применением ilpy (`syr2k_ilp_async`) и pluto (`syr2k_pluto`), запускались с количеством нитей OpenMP, равным 1, 2, 4, 6, 8. Значения параметров `N` и `M` заданы равными 1536. Графики ускорения относительно запуска последовательного варианта `syr2k_vanilla`, а также сравнение двух параллельных вариантов приведены на рисунке 4.13. Подробная статистика запусков приведена в приложении Г.5.

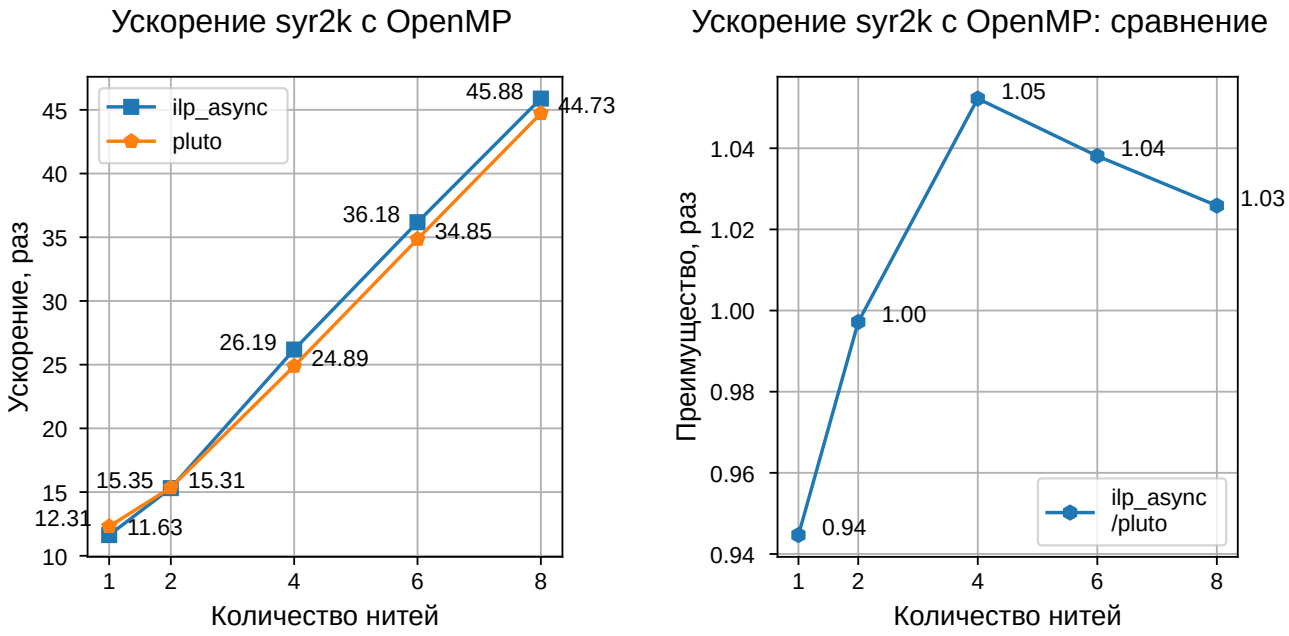


Рисунок 4.13 — Ускорение syr2k с OpenMP

Параллельный вариант `syr2k_ilp_async` показывает лучшую производительность в параллельных запусках. Наибольшее преимущество перед pluto достигается при запуске в четырех нитях и составляет 5%.

4.4.2 Распараллеливание с MPI

Вычисление пространственных и временных отображений осуществляется запуском ilpy с параметром `--arrays`.

```
ilpy syr2k.c --arrays --async --out-fn-suffix aarrays > syr2k.c-aarrays.txt
   2>&1
```

Вычисленные ilpy аффинные отображения имеют вид:

$$\theta_{S_0} = \begin{bmatrix} 0 \end{bmatrix}; \quad \pi_{S_0} = \begin{bmatrix} i \\ -j + N - 1 \end{bmatrix}; \quad \theta_{S_1} = \begin{bmatrix} k + 1 \end{bmatrix}; \quad \pi_{S_1} = \begin{bmatrix} i \\ -j + N - 1 \end{bmatrix};$$

$$\eta_C = \begin{bmatrix} i0 \\ -i1 + N - 1 \end{bmatrix}; \quad \eta_A = \begin{bmatrix} i0 \\ -i0 + N - 1 \end{bmatrix}; \quad \eta_B = \begin{bmatrix} i0 \\ -i0 + N - 1 \end{bmatrix}.$$

В соответствии с ними генерируется параллельная программа, представленная на листинге Г.7.

Расписание совпадает с тем, что было найдено для варианта OpenMP. Размещение вычислений отличается только во втором компоненте. Свойство вперед направленных коммуникаций сохраняется. Найденное размещение массивов `A`, `B` и `C` можно интерпретировать следующим образом: строка матрицы с индексом `i` располагается на виртуальном процессоре `i`. Второй компонент размещения данных игнорируется, так как пространство виртуальных процессоров предполагается одномерным. На листинге Г.4 представлена информация о найденных совместно размещении вычислений и размещении данных: для семи из девяти доступов к памяти удается ограничить $d^{\rho}_{A_{S_j,i}}$ нулем, что означает отсутствие для данных доступов необходимости организации удаленной записи или удаленного чтения, поскольку совпадает индекс виртуального процессора-вычислителя с индексом виртуального процессора-владельца.

На листинге Г.8 представлена программа syr2k с асинхронным параллелизмом, дополненная конструкциями для информационного обмена в рамках стандарта MPI. Реализация операций удаленного чтения выполняется перед параллельным циклом при помощи разработанной библиотеки макросов blockdist.h. Пересылка строк матриц выполняется применением пары макросов `line_Q_read_send` и `line_Q_read_receive`, при этом заданы макросы `eta_A_i` и `eta_B_i`, определяющие размещение элемента массива, принадлежащего строке `i`. Вычисления завершаются сбором результата в процессе с рангом 0 с помощью разработанной функции `collect_matrix_rows_double`.

Транслятор pluto выдает программный код (листинг Г.13) в соответствии со следующими пространственно-временными отображениями (листинг Г.12):

$$\Phi_{S_0} = \begin{bmatrix} 0 & 0 & 0 & i & j & 0 \end{bmatrix}^T; \quad \Phi_{S_1} = \begin{bmatrix} 1 & 0 & 0 & i & j & k \end{bmatrix}^T.$$

На листинге А.30 приведен файл сборки тестового приложения syr2k_mpi для MPI. Параллельные варианты программы `syr2k_vanilla`, полученные применением ilpy (`syr2k_ilp_arrays_async`) и pluto (`syr2k_pluto_mpi`), запускались с количеством процессов MPI, равным 1, 2, 4, 6, 8 в трех вариантах запуска: все процессы на одной машине (суффикс _one), процессы распределены поровну между двумя машинами (суффикс `_two`), количество машин равно количеству процессов (суффикс _eq). В каждом процессе MPI поддерживалась единая рабочая нить. Значения параметров `N` и `M` заданы равными 1536. Графики ускорения относительно запуска последовательного варианта `syr2k_vanilla`, а также сравнение двух параллельных вариантов приведены на рисунке 4.14.

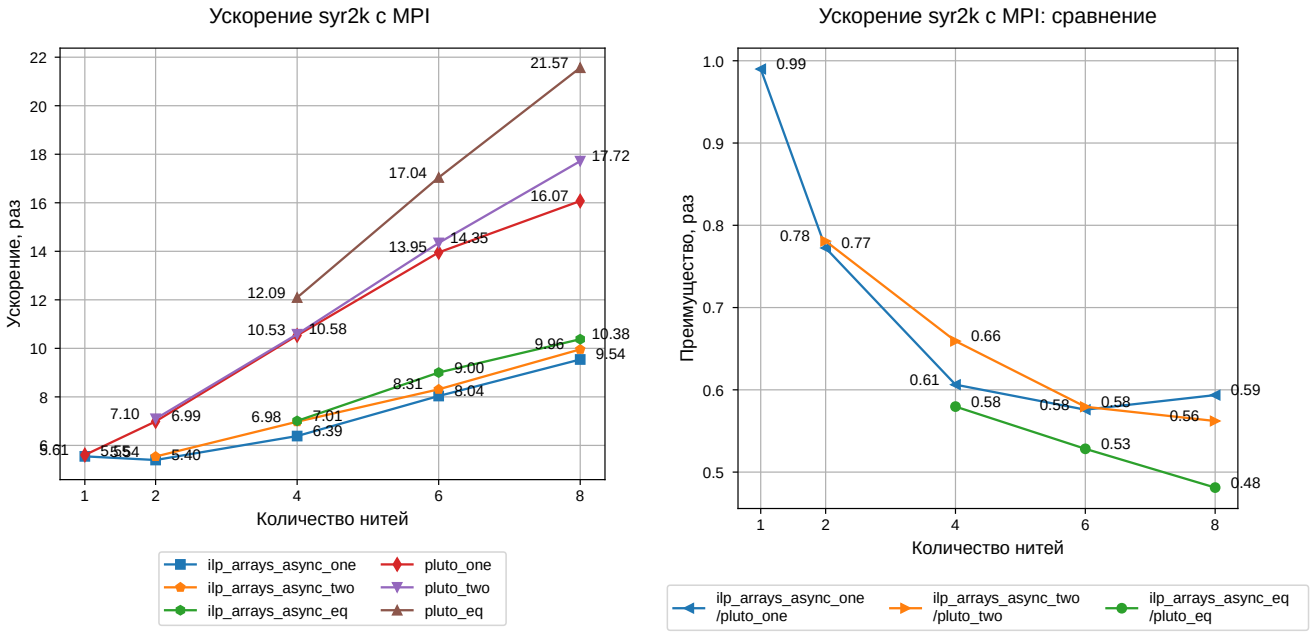


Рисунок 4.14 — Ускорение syr2k с MPI

Все решения позволили получить ускорение вычислений. Параллельный вариант `syr2k_ilp_arrays_async` показывает меньшую производительность во всех запусках. Проигрыш в производительности составляет до 2 раз в зависимости от количества процессов. Причиной является меньшая загруженность процессоров вычислениями и большая вовлеченность в информационный обмен, поскольку входные данные не дублированы во всех процессах. Подробная статистика запусков приведена в приложении Г.6.

На рисунке 4.15 представлены графики, иллюстрирующие долю времени вычислений во всех запусках, а также приводится сравнение соответственных вариантов, полученных применением ilpy и pluto. Решения pluto во всех параллельных запусках демонстрируют лучшую загруженность процессоров.

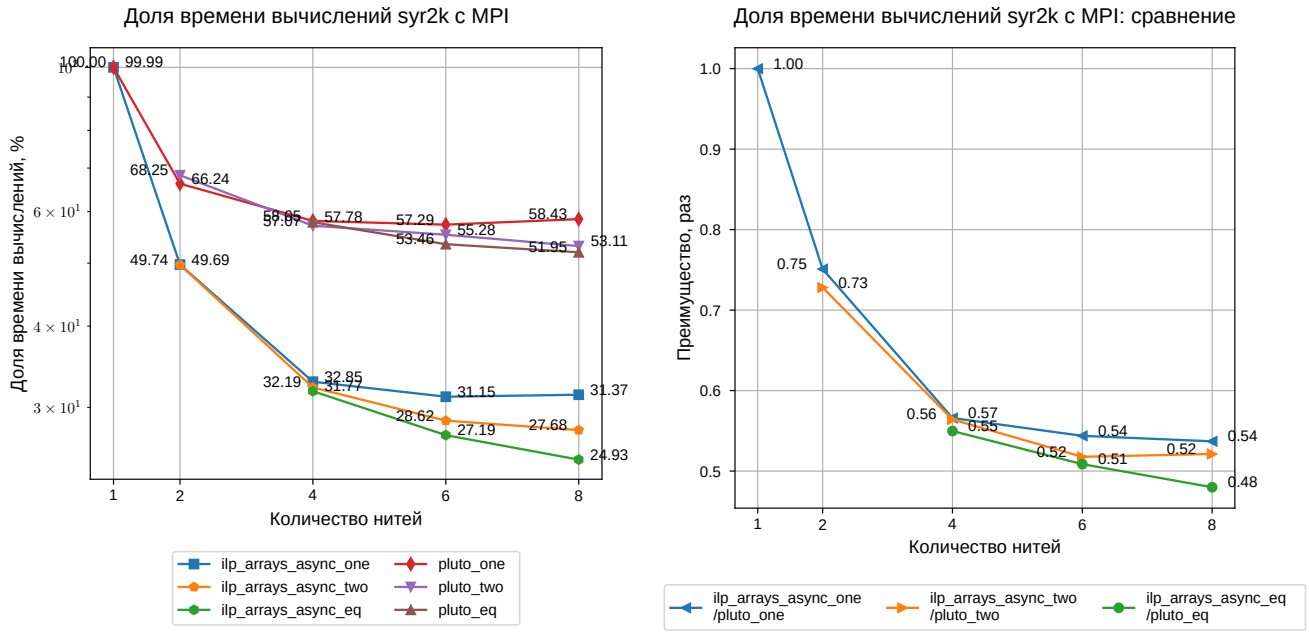


Рисунок 4.15 — Доля времени вычислений в запусках syr2k с MPI

На рисунке 4.16 проиллюстрировано распределение времени исполнения кода между вычислениями, информационным обменом и барьерной синхронизацией. Во всех запусках решения pluto демонстрируют меньшую загруженность накладными расходами, особенно в процессах с наибольшим рангом. Это справедливо при уменьшении количества процессов с восьми до двух.

4.5 Алгоритм Флойда–Уоршалла

4.5.1 Распараллеливание с OpenMP

Рассматривается программа из набора тестов polybench [110], выполняющая вычисления «на месте», то есть результат окажется в той же области памяти, что и входные данные (функция `floyd_vanilla`, листинг Д.1). Разработанный

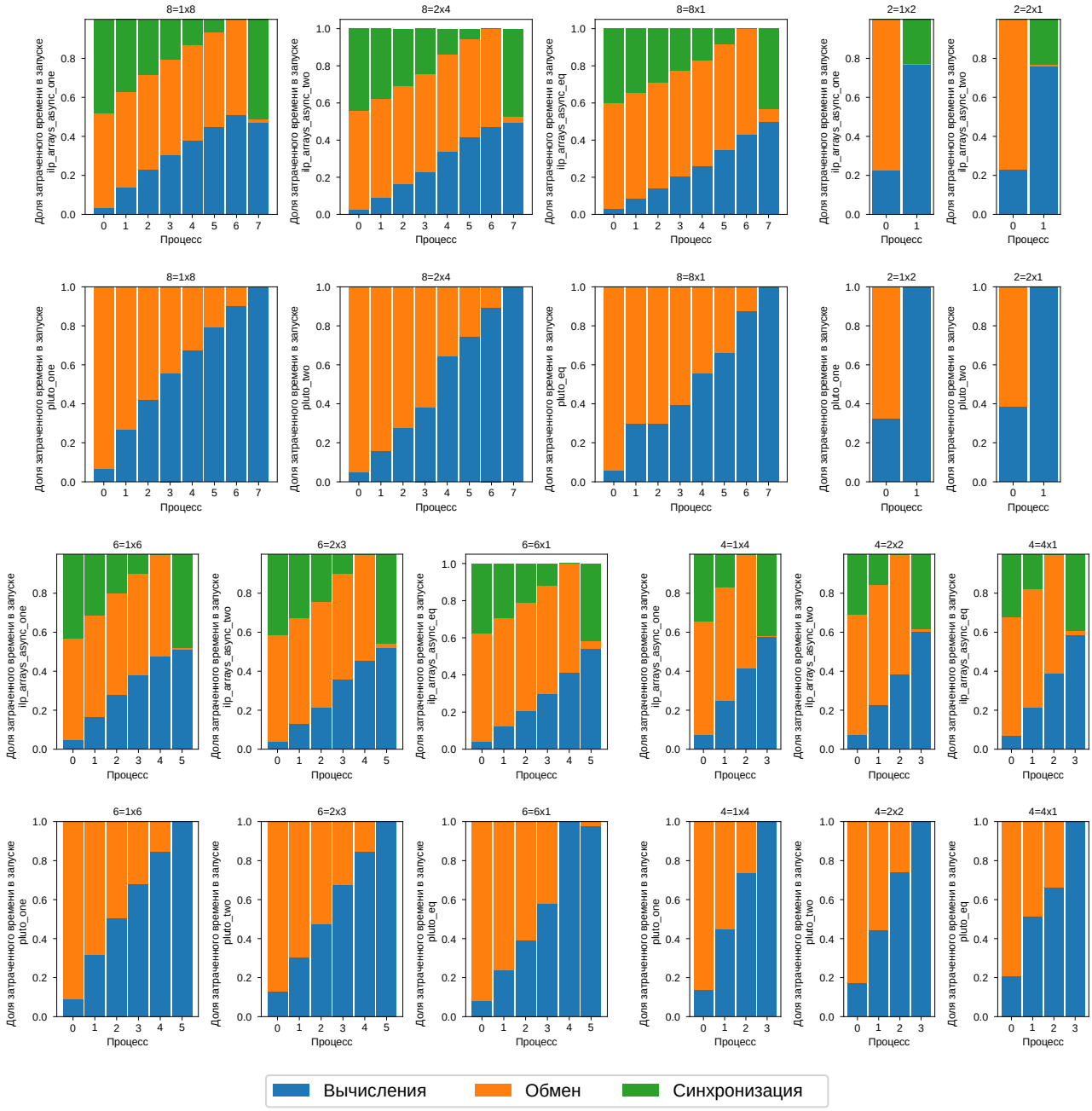


Рисунок 4.16 — Разнородная нагрузка при запуске параллельных вариантов syr2k с MPI

транслятор ilpy принимает на вход фрагмент кода функции `floyd_vanilla` (листинг Д.1), обрамленный директивами препроцессора `#pragma scop` и `#pragma endscop` в упрощенном виде: исключается операция минимума для обеспечения совместимости с используемыми библиотеками. На этапах постобработки она будет возвращена простой текстовой заменой.

```
ilpy floyd.c > floyd.c-deps.txt 2>&1
```

Внешняя переменная `N` трактуется как размер задачи и его вес устанавливается как значение по умолчанию, равное 100. Вычисленные ilpy аффинные отображения имеют вид:

$$\theta_{S_0} = \begin{bmatrix} k \\ i+j \end{bmatrix}; \quad \pi_{S_0} = \begin{bmatrix} j \end{bmatrix}.$$

Найдено двумерное расписание для удовлетворения всех информационных зависимостей. Размещение вычислений не обладает свойством вперед направленных коммуникаций. На листинге Д.2 представлена информация о найденных расписании и размещении вычислений соответственно.

Первый компонент найденного расписания позволяет ограничить $d^{\tau}_{R_i}$ для всех семи соответствующих ребер в обобщенном графе зависимостей постоянными величинами, а второй компонент — для оставшихся четырех ребер. Найденное размещение вычислений позволяет ограничить $d^{\rho}_{R_i}$ постоянной величиной для девяти из одиннадцати ребер в обобщенном графе зависимостей.

На листинге Д.4 представлена программа floyd с синхронным параллелизмом, генерируемая ilpy. Итоговый вариант с постобработкой и расстановкой директив OpenMP представлен на листинге Д.5.

Транслятор pluto выдает программный код (листинг Д.9) в соответствии со следующими пространственно-временными отображениями (листинг Д.8):

$$\Phi_{S_0} = \begin{bmatrix} k \\ i+j \\ j \end{bmatrix}.$$

Результат постобработки решения pluto представлен на листинге Д.10.

На листинге А.25 приведен файл сборки тестового приложения floyd для OpenMP. Параллельные варианты программы `floyd_vanilla`, полученные применением ilpy (`floyd_ilp_sync`) и pluto (`floyd_pluto`), запускались с количеством нитей OpenMP, равным 1, 2, 4, 6, 8. Значение параметра `N` задано равным 1024. Графики ускорения относительно запуска последовательного варианта `floyd_vanilla`, а также сравнение двух параллельных вариантов приведены на рисунке 4.17. Подробная статистика запусков приведена в приложении Д.5.

Параллельные варианты `floyd_ilp_sync` и `floyd_pluto` показывают одинаковую производительность (разница в измерениях не превышает 1%), так как материализуют одни и те же аффинные отображения. Ускорения вычислений не достигается из-за дорогого доступа к матрице вдоль ее диагоналей.

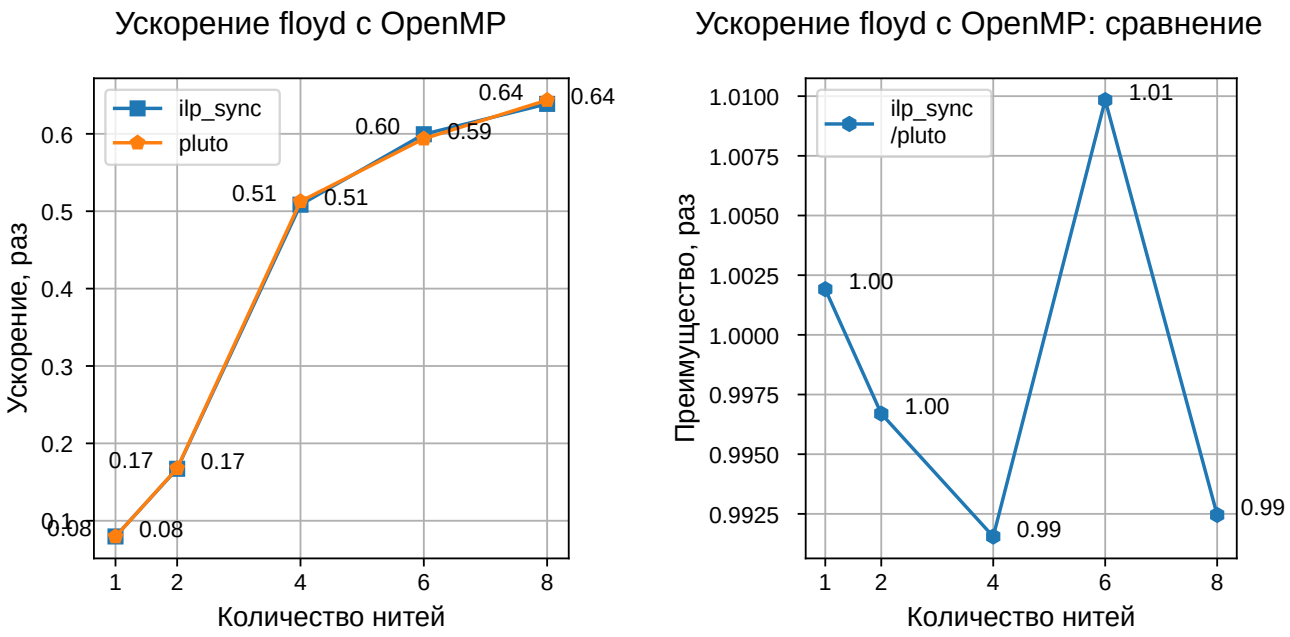


Рисунок 4.17 — Ускорение floyd с OpenMP

4.5.2 Распараллеливание с MPI

Вычисление пространственных и временных отображений осуществляется запуском ilpy с параметром `--arrays`.

```
ilpy floyd.c --arrays > floyd.c-arrays.txt 2>&1
```

Вычисленные ilpy аффинные отображения имеют вид:

$$\theta_{S_0} = \begin{bmatrix} k \\ i+j \end{bmatrix}; \quad \pi_{S_0} = \begin{bmatrix} j \end{bmatrix}; \quad \eta_A = \begin{bmatrix} i1 \end{bmatrix}.$$

В соответствии с ними генерируется параллельная программа, представленная на листинге Д.6.

Расписание и размещение вычислений совпадает с тем, что было найдено для варианта OpenMP. Найденное размещение массива `A` можно интерпретировать следующим образом: столбец матрицы `A` с индексом `i` располагается на виртуальном процессоре `i`.

На листинге Д.3 представлена информация о найденных совместно размещении вычислений и размещении данных: для трех из четырех доступов к памяти удается ограничить $d^{\rho}_{A_{S_j,i}}$ нулем, что означает отсутствие для данных доступов необходимости организации удаленной записи или удаленного чтения, поскольку

совпадает индекс виртуального процессора-вычислителя с индексом виртуального процессора-владельца.

На листинге Д.7 представлена программа floyd с синхронным параллелизмом, дополненная конструкциями для информационного обмена в рамках стандарта MPI. Реализация операций удаленного чтения выполняется перед параллельным циклом при помощи разработанной библиотеки макросов blockdist.h. Здесь пересылка подстолбца матрицы выполняется с помощью `col_Q_read_vp_rev`, при этом задан буфер для пересылки `buf_A`. Вычисления завершаются сбором результата в процессе с рангом 0 с помощью разработанной функции `collect_matrix_cols_double`.

Транслятор pluto выдает программный код (листинг Д.12) в соответствии со следующими пространственно-временными отображениями (листинг Д.11):

$$\Phi_{S_0} = \begin{bmatrix} k \\ i+j \\ 0 \\ j \end{bmatrix}.$$

На листинге А.31 приведен файл сборки тестового приложения floyd_mpi для MPI. Параллельные варианты программы `floyd_vanilla`, полученные применением ilpy (`floyd_ilp_arrays`) и pluto (`floyd_pluto_mpi`), запускались с количеством процессов MPI, равным 1, 2, 4, 6, 8 в трех вариантах запуска: все процессы на одной машине (суффикс `_one`), процессы распределены поровну между двумя машинами (суффикс `_two`), количество машин равно количеству процессов (суффикс `_eq`). В каждом процессе MPI поддерживалась единая рабочая нить. Значение параметра `N` задано равным 1024. Графики ускорения относительно запуска последовательного варианта `floyd_vanilla`, а также сравнение двух параллельных вариантов приведены на рисунке 4.18.

Никакие решения не позволили получить ускорение вычислений. Параллельный вариант `floyd_ilp_arrays` показывает лучшую производительность во всех запусках. Наибольшее преимущество перед pluto достигается при запуске в шести процессах и составляет 3,92 раза для запуска на одной машине и 2,58 раз для запуска на двух машинах. Подробная статистика запусков приведена в приложении Д.6. На рисунке 4.19 представлены графики, иллюстрирующие долю времени вычислений во всех запусках, а также приводится сравнение соответственных вариантов, полученных применением ilpy и pluto. Решения pluto во всех параллельных запусках демонстрируют худшую загруженность процессоров.

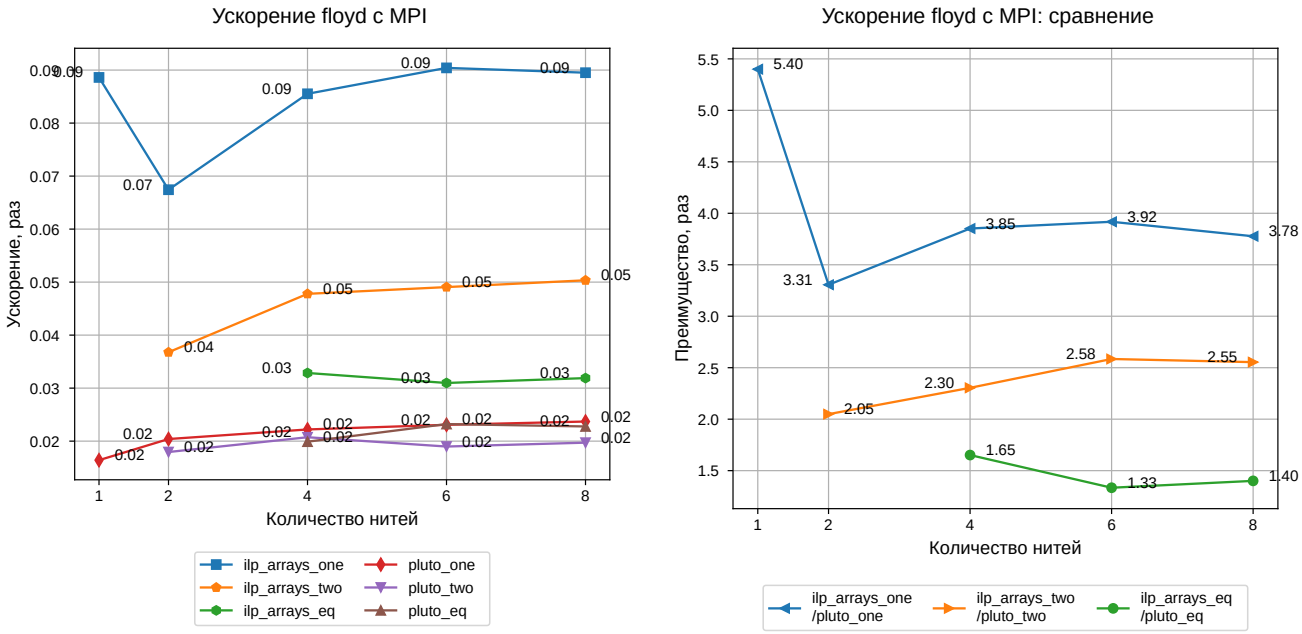


Рисунок 4.18 — Ускорение floyd с MPI

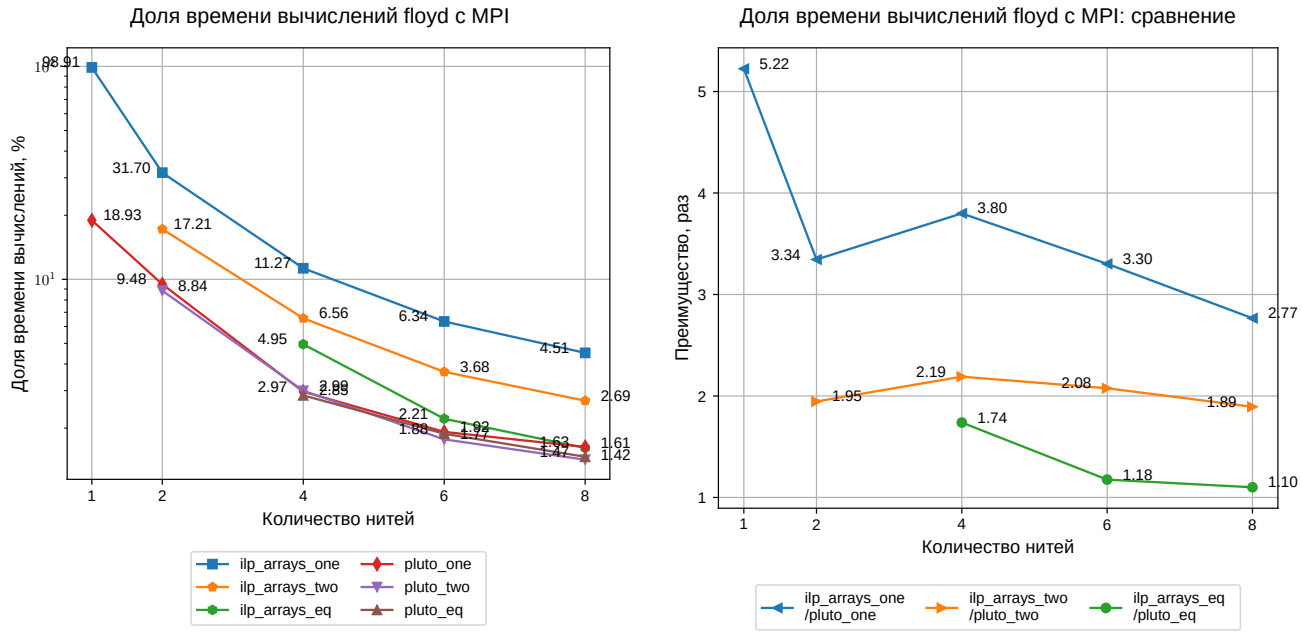


Рисунок 4.19 — Доля времени вычислений в запусках floyd с MPI

На рисунке 4.20 проиллюстрировано распределение времени исполнения кода между вычислениями, информационным обменом и барьерной синхронизацией. Во всех запусках решения pluto выглядят сбалансированными, но демонстрируют меньшую загруженность вычислениями и более весомые накладные расходы. Это справедливо при уменьшении количества процессов с восьми до двух.

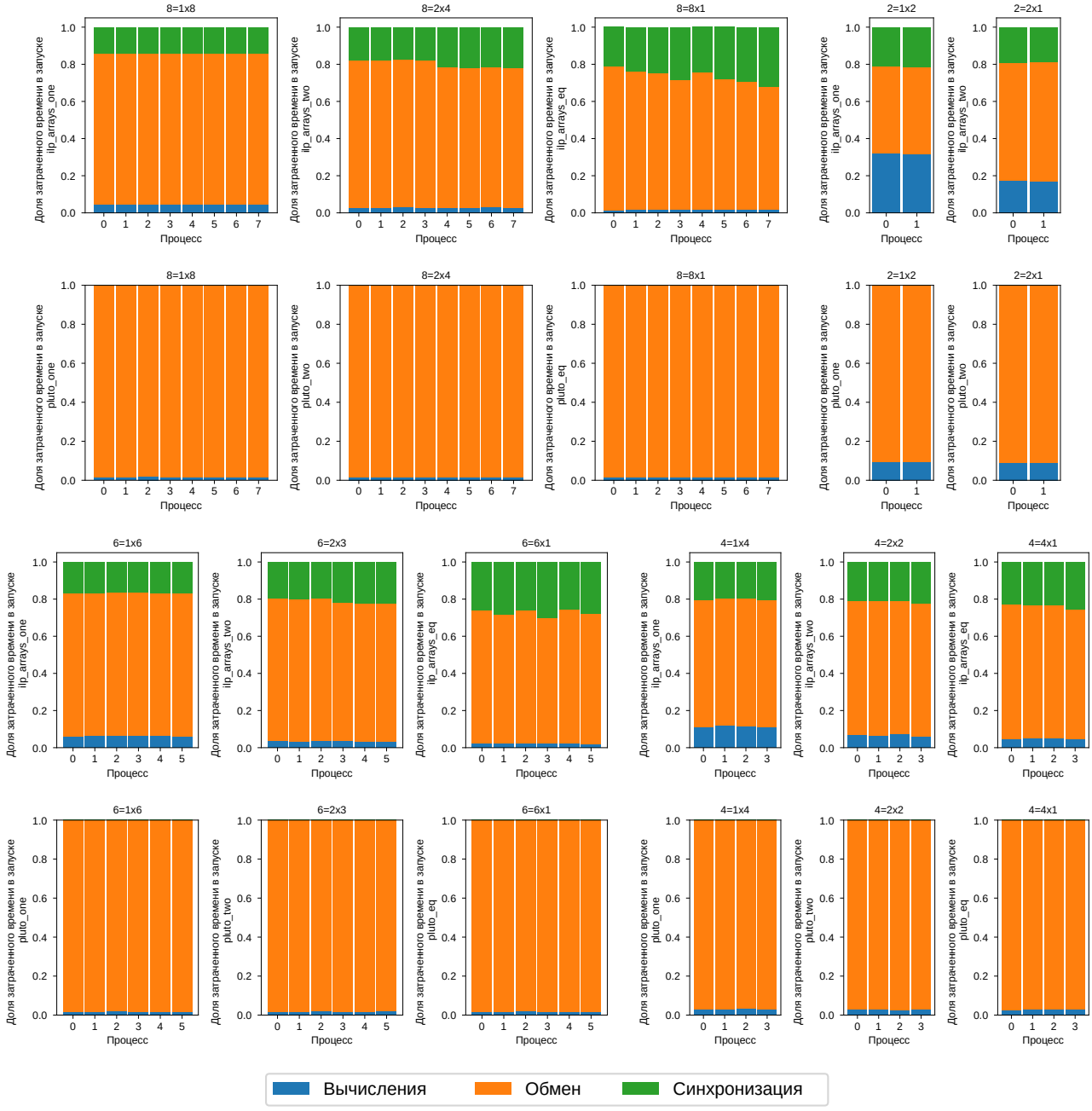


Рисунок 4.20 — Разнородная нагрузка при запуске параллельных вариантов floyd с MPI

4.6 Процедура gramschmidt

4.6.1 Распараллеливание с OpenMP

Рассматривается программа из набора тестов polybench [110], выполняющая вычисления «на месте», то есть результат окажется в той же области памяти, что и входные данные (функция `gramschmidt_vanilla`, листинг Е.1). Разработанный транслятор ilpy принимает на вход фрагмент кода функции `gramschmidt_vanilla` (листинг Е.1), обрамленный директивами препроцессора `#pragma scop` и `#pragma endscop` в упрощенном виде: исключается операция взятия квадратного корня для обеспечения совместимости с используемыми библиотеками. На этапах постобработки она будет возвращена простой текстовой заменой.

```
ilpy gramschmidt.c --cloog-f 1 --ilp-col-ub 2 --iocp-tm-lim 30 --param-
   weight M=101,N=100 > gramschmidt.c-deps.txt 2>&1
```

Внешние переменные `M` и `N` трактуются как размер задачи и их оценки передаются с параметром `--param-weight`. С целью сокращения времени работы решателя glpk при нахождении размещений, устанавливается ограничение на значение переменных, равное 2. Также устанавливается лимит времени работы решателя в 30 секунд. Вычисленные ilpy аффинные отображения имеют вид:

$$\theta_{S_0} = \begin{bmatrix} 5k \\ 0 \end{bmatrix}; \qquad \pi_{S_0} = \begin{bmatrix} k+N-1 \end{bmatrix}; \quad \theta_{S_1} = \begin{bmatrix} 5k+1 \\ i \end{bmatrix}; \quad \pi_{S_1} = \begin{bmatrix} k+N-1 \end{bmatrix};$$

$$\theta_{S_2} = \begin{bmatrix} 5k+2 \\ 0 \end{bmatrix}; \quad \pi_{S_2} = \begin{bmatrix} k+N-1 \end{bmatrix}; \quad \theta_{S_3} = \begin{bmatrix} 5k+3 \\ 0 \end{bmatrix}; \quad \pi_{S_3} = \begin{bmatrix} k+i+N-1 \end{bmatrix};$$

$$\theta_{S_4} = \begin{bmatrix} 5k+3 \\ 0 \end{bmatrix}; \quad \pi_{S_4} = \begin{bmatrix} j+N-2 \end{bmatrix}; \quad \theta_{S_5} = \begin{bmatrix} 5k+4 \\ i \end{bmatrix}; \quad \pi_{S_5} = \begin{bmatrix} j+i+N-1 \end{bmatrix};$$

$$\theta_{S_6} = \begin{bmatrix} 5k+5 \\ 0 \end{bmatrix}; \quad \pi_{S_6} = \begin{bmatrix} j+i+N-1 \end{bmatrix}.$$

Найдено двумерное расписание для удовлетворения всех информационных зависимостей. Размещение вычислений не обладает свойством вперед направленных коммуникаций. На листинге Е.2 представлена информация о найденных

расписании и размещении вычислений соответственно. Первый компонент найденного расписания позволяет ограничить $d^{\tau}_{R_i}$ для всех двадцати восьми соответствующих ребер в обобщенном графе зависимостей постоянными величинами, а второй компонент — для оставшихся шести ребер. Найденное размещение вычислений позволяет ограничить $d^{\rho}_{R_i}$ постоянной величиной для двадцати пяти из тридцати четырех ребер в обобщенном графе зависимостей.

На листинге Е.4 представлена программа gramschmidt с синхронным параллелизмом, генерируемая ilpy. Итоговый вариант с постобработкой и расстановкой директив OpenMP представлен на листинге Е.5.

Транслятор pluto выдает программный код (листинг Е.9) в соответствии со следующими пространственно-временными отображениями (листинг Е.8):

$$\Phi_{S_0} = \begin{bmatrix} 1 & k & 1 & 0 & 1 & 0 \end{bmatrix}^T; \qquad \Phi_{S_1} = \begin{bmatrix} 1 & k & 2 & i & 1 & 0 \end{bmatrix}^T;$$
$$\Phi_{S_2} = \begin{bmatrix} 1 & k & 3 & 0 & 1 & 0 \end{bmatrix}^T; \qquad \Phi_{S_3} = \begin{bmatrix} 1 & k & 4 & i & 1 & 0 \end{bmatrix}^T;$$
$$\Phi_{S_4} = \begin{bmatrix} 0 & k & 0 & j & 1 & 0 \end{bmatrix}^T; \qquad \Phi_{S_5} = \begin{bmatrix} 1 & k & 5 & j & 0 & i \end{bmatrix}^T;$$
$$\Phi_{S_6} = \begin{bmatrix} 1 & k & 5 & j & 1 & i \end{bmatrix}^T.$$

Результат постобработки решения pluto представлен на листинге Е.10.

На листинге А.26 приведен файл сборки тестового приложения gramschmidt для OpenMP. Параллельные варианты `gramschmidt_vanilla`, полученные применением ilpy (`gramschmidt_ilp_sync`) и pluto (`gramschmidt_pluto`), запускались с количеством нитей OpenMP, равным 1, 2, 4, 6, 8. Значения параметров: `N` задано 1024, `M` задано 2048. Графики ускорения относительно запуска последовательного варианта `gramschmidt_vanilla`, а также сравнение двух параллельных вариантов приведены на рисунке 4.21. Подробная статистика запусков приведена в приложении Е.5.

Параллельный вариант `gramschmidt_ilp_sync` показывает лучшую производительность во всех запусках. Наибольшее преимущество перед pluto достигается при запуске в одной нити и составляет 2,31 раза. Это свидетельствует об улучшении производительности за счет улучшения локальности использования данных даже без параллелизма.

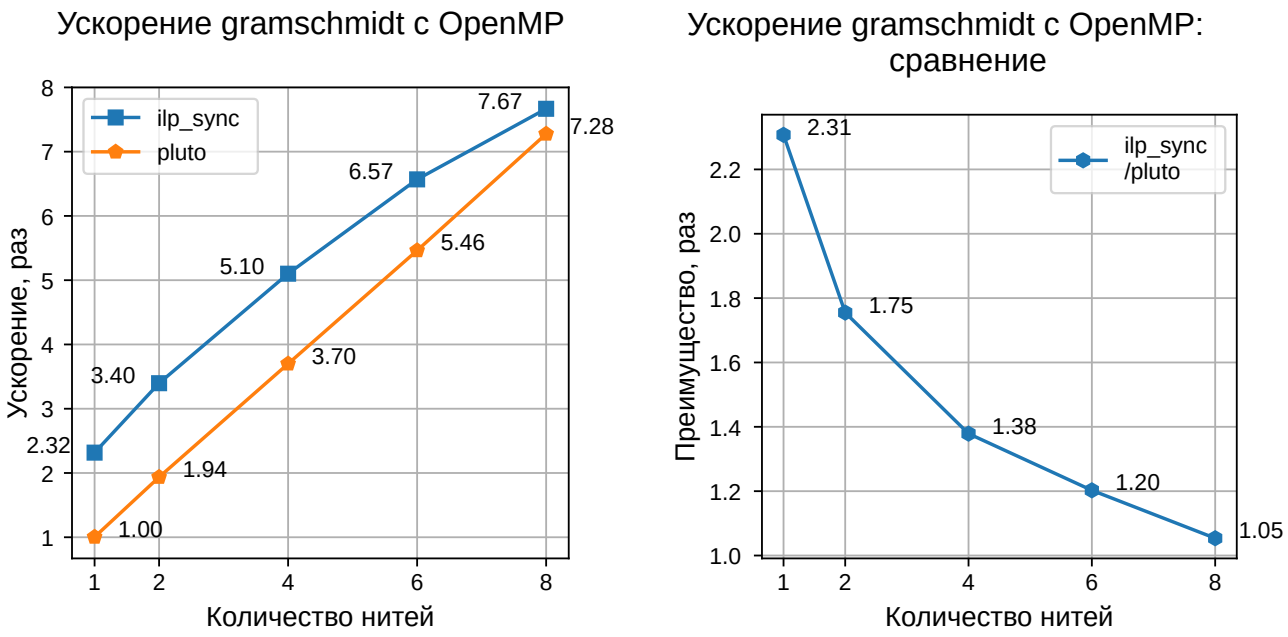


Рисунок 4.21 — Ускорение gramschmidt с OpenMP

4.6.2 Распараллеливание с MPI

Вычисление пространственных и временных отображений осуществляется запуском ilpy с параметром `--arrays`.

```
ilpy gramschmidt.c --arrays --cloog-f 1 --ilp-col-ub 2 --param-weight M=101,
   N=100 > gramschmidt.c-arrays.txt 2>&1
```

Вычисленные ilpy аффинные отображения имеют вид:

$$\theta_{S_0} = \begin{bmatrix} 5k \\ 0 \end{bmatrix}; \quad \pi_{S_0} = \begin{bmatrix} k \end{bmatrix}; \quad \theta_{S_1} = \begin{bmatrix} 5k+1 \\ i \end{bmatrix}; \quad \pi_{S_1} = \begin{bmatrix} k+1 \end{bmatrix};$$

$$\theta_{S_2} = \begin{bmatrix} 5k+2 \\ 0 \end{bmatrix}; \quad \pi_{S_2} = \begin{bmatrix} k+1 \end{bmatrix}; \quad \theta_{S_3} = \begin{bmatrix} 5k+3 \\ 0 \end{bmatrix}; \quad \pi_{S_3} = \begin{bmatrix} k+i \end{bmatrix};$$

$$\theta_{S_4} = \begin{bmatrix} 5k+3 \\ 0 \end{bmatrix}; \quad \pi_{S_4} = \begin{bmatrix} j+1 \end{bmatrix}; \quad \theta_{S_5} = \begin{bmatrix} 5k+4 \\ i \end{bmatrix}; \quad \pi_{S_5} = \begin{bmatrix} j+1 \end{bmatrix};$$

$$\theta_{S_6} = \begin{bmatrix} 5k+5 \\ 0 \end{bmatrix}; \quad \pi_{S_6} = \begin{bmatrix} j+1 \end{bmatrix};$$

$$\eta_A = \begin{bmatrix} i1+1 \end{bmatrix}; \quad \eta_Q = \begin{bmatrix} i1+1 \end{bmatrix}; \quad \eta_R = \begin{bmatrix} i1+1 \end{bmatrix}; \quad \eta_{nrm} = [\quad].$$

В соответствии с ними генерируется параллельная программа, представленная на листинге Е.6. Расписание совпадает с тем, что было найдено для варианта OpenMP. Размещение вычислений не обладает свойством вперед направленных коммуникаций. Найденное размещение массивов `A`, `Q`, `R` можно интерпретировать следующим образом: столбец матрицы с индексом `i` располагается на виртуальном процессоре `i+1`. Переменная `nrm` является локальной. На листинге Е.3 представлена информация о найденных совместно размещении вычислений и размещении данных: для десяти из девятнадцати доступов к памяти удается ограничить $d^{\rho}_{A_{S_j},i}$ нулем, что означает отсутствие для данных доступов необходимости организации удаленной записи или удаленного чтения, поскольку совпадает индекс виртуального процессора-вычислителя с индексом виртуального процессора-владельца. На листинге Е.7 представлена программа gramschmidt с синхронным параллелизмом, дополненная конструкциями для информационного обмена в рамках стандарта MPI. Реализация операций удаленного доступа выполняется при помощи разработанной библиотеки макросов blockdist.h. Вычисления завершаются сбором результата в процессе с рангом 0 с помощью разработанной функции `collect_matrix_cols_double`.

Транслятор pluto завершает работу с ошибкой, поэтому его результаты приравниваются к результатам исходной последовательной программы. На листинге А.32 приведен файл сборки тестового приложения gramschmidt_mpi для MPI. Параллельный вариант `gramschmidt_vanilla`, полученный применением ilpy (`gramschmidt_ilp_arrays`), запускался с количеством процессов MPI, равным 1, 2, 4, 6, 8 в трех вариантах запуска: все процессы на одной машине (суффикс `_one`), процессы распределены поровну между двумя машинами (суффикс `_two`), количество машин равно количеству процессов (суффикс `_eq`). В каждом процессе MPI поддерживалась единая рабочая нить. Значения параметров: `N` задано 1024, `M` задано 2048. Графики ускорения относительно запуска последовательного варианта `gramschmidt_vanilla` и графики, иллюстрирующие долю времени вычислений во всех запусках, приведены на рисунке 4.22.

Не все решения позволили получить ускорение вычислений. Параллельный вариант `gramschmidt_ilp_arrays` показывает лучшую производительность только в запусках на одной машине. Подробная статистика запусков приведена в приложении Б.6. На рисунке 4.23 проиллюстрировано распределение времени исполнения кода между вычислениями, информационным обменом и барьерной синхронизацией.

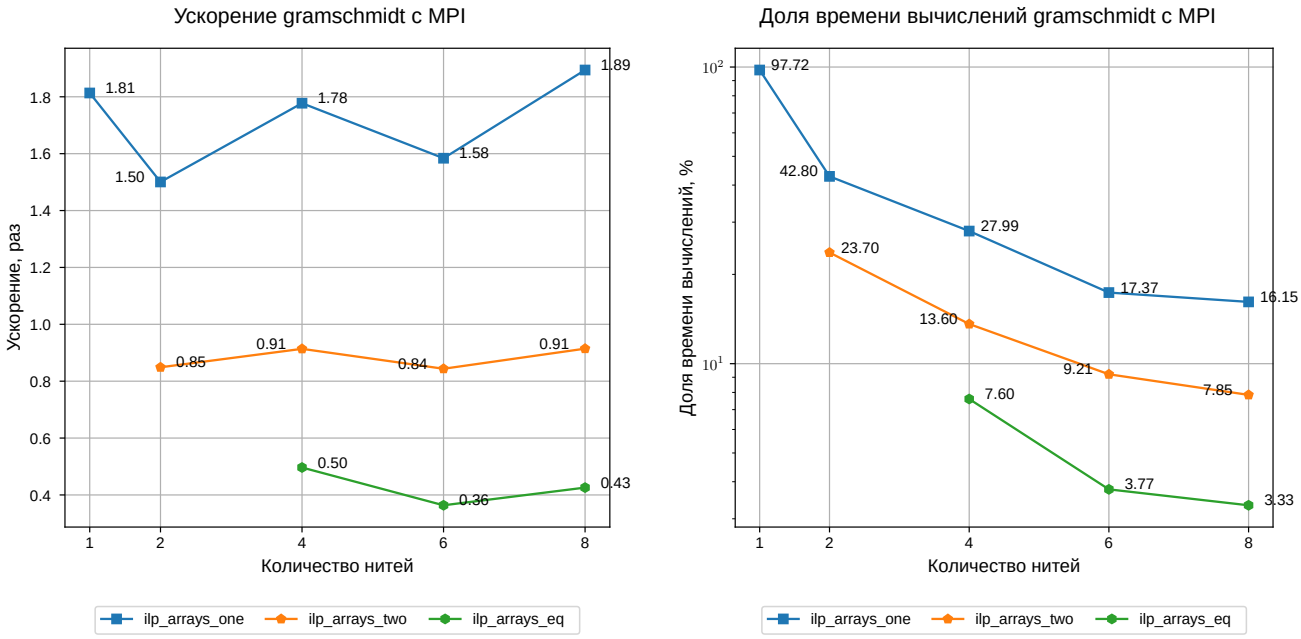


Рисунок 4.22 — Ускорение и доля времени вычислений в запусках gramschmidt с MPI

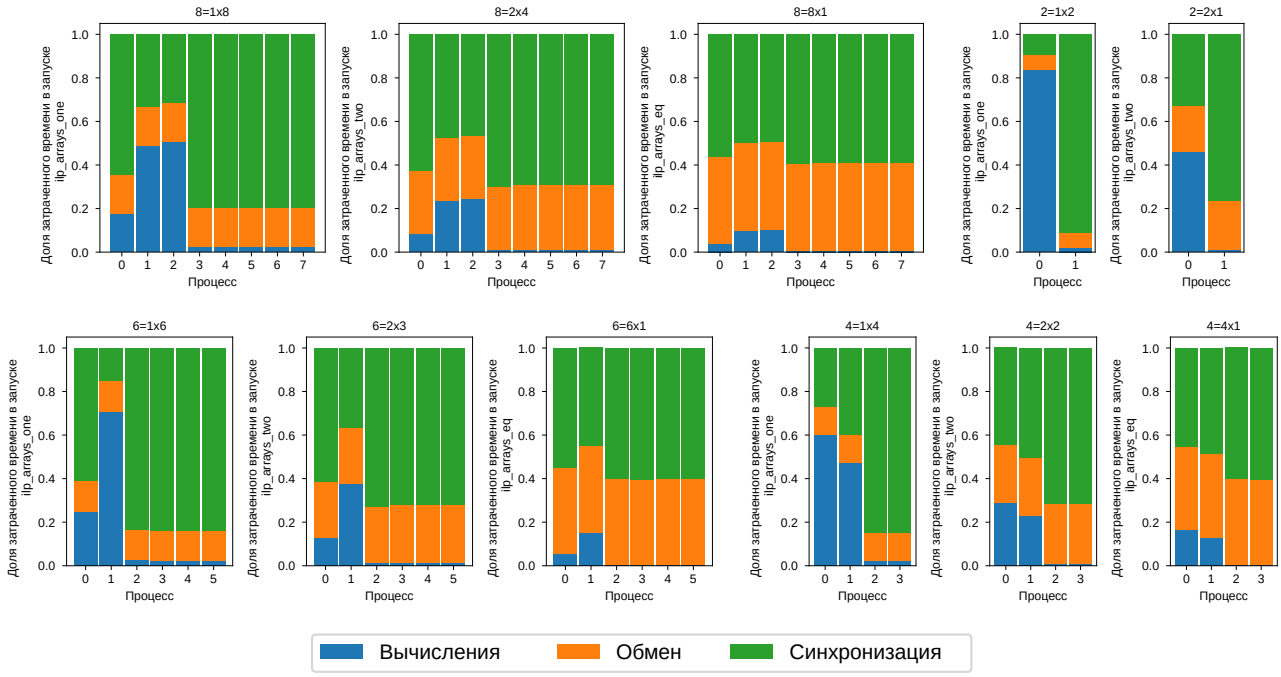


Рисунок 4.23 — Разнородная нагрузка при запуске параллельных вариантов gramschmidt с MPI

Причиной низкой производительности в распределенных запусках является недостаточная загрузка процессоров вычислениями — нагрузка информационного обмена слишком велика. Во всех запусках решения ilpy демонстрируют меньшую загруженность вычислениями и более весомые накладные расходы в процессах с большим рангом, и большую загруженность вычислениями и менее весомые накладные расходы в процессах с меньшим рангом. Это справедливо при уменьшении количества процессов с восьми до двух.

4.7 Общие выводы по главе

На рисунке 4.24 проиллюстрированы результаты запуска параллельных программ в 8 нитях OpenMP и 8 процессах MPI (одна нить на процесс). По результатам экспериментов видно, что самыми удачными для распараллеливания оказались программы lu и syr2k, поскольку для них удалось достичь заметного прироста в производительности как с OpenMP, так и с MPI. Практически значимые результаты были достигнуты как применением разработанного транслятора ilpy, так и современного транслятора pluto. Распараллеливание программ atax (в исходном варианте atax_p) и floyd привело лишь к замедлению вычислений за счет накладных расходов на поддержку параллелизма. Выигрыш в быстродействии atax, который дает pluto в варианте для OpenMP, составляет всего 13% при запуске в 8 нитей. Для программы gramschmidt удалось достичь практически значимого ускорения более, чем 7 раз на 8 нитях при распараллеливании с OpenMP, однако распараллеливание с MPI не увенчалось успехом: вариант ilpy дал ускорение всего на 89% при запуске на одной машине, а транслятор pluto не смог завершить работу корректно.

Пусть S_p^{tool} — ускорение, полученное для запуска в p процессах MPI (или нитях OpenMP) программы, порожденной средством $tool$. Преимущество ilpy перед pluto оценивалось по формуле $(S_8^{ilpy}/S_8^{pluto} - 1) \cdot 100\%$. Здесь ускорение может быть заменено на эффективность при том же количестве параллельно работающих процессов (нитей), что не повлияет на результат. Полученное значение позволяет оценить преимущество не только в ускорении, но и в эффективности. На рисунке 4.25 проиллюстрировано сравнение эффективности параллельных вариантов программ, полученных применением ilpy и pluto. Для программы lu

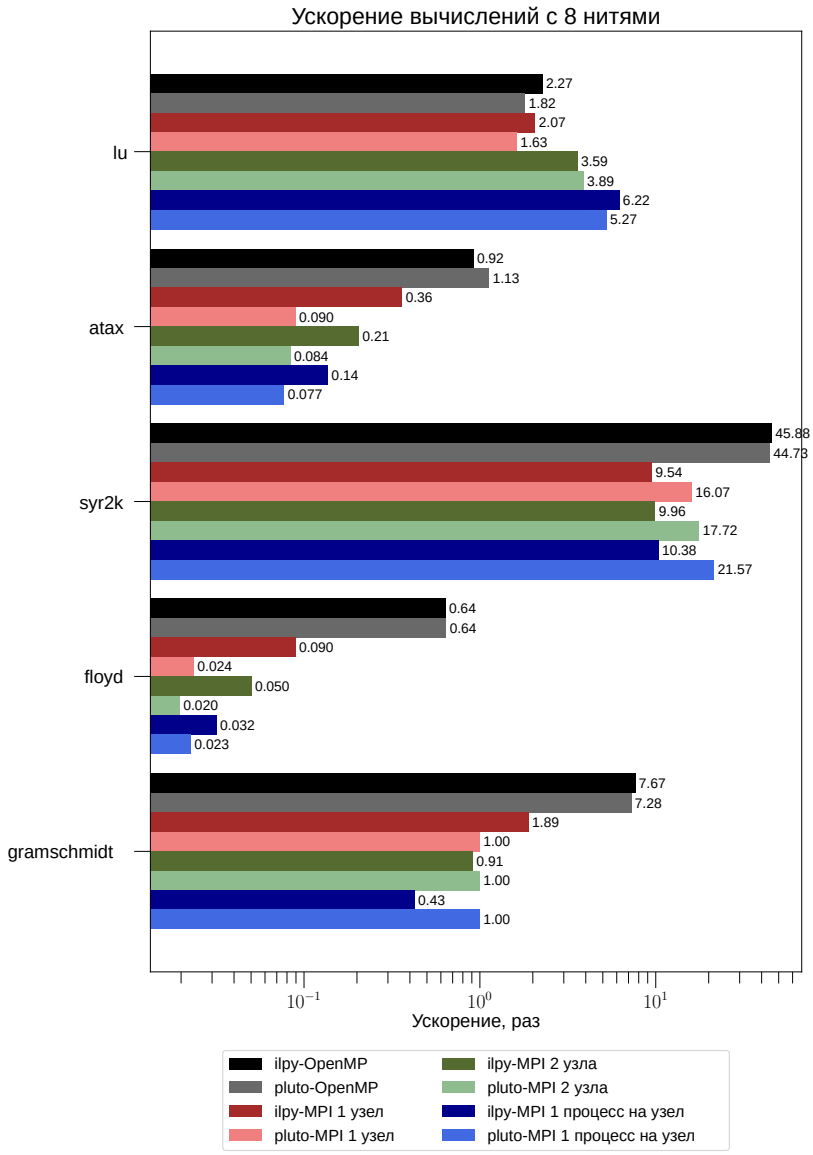


Рисунок 4.24 — Ускорение линейных программ при распараллеливании в 8 нитях (процессах)

преимущество остается за ilpy почти во всех запусках, кроме запуска с MPI на двух машинах. Для программы atax (в варианте atax_p) ilpy дает на 18% худшие результаты в варианте OpenMP, но во всех запусках с MPI показывает преимущество перед pluto. Для программы syr2k ilpy дает преимущество в 2,59% с OpenMP, однако результаты MPI оказываются приблизительно вдвое медленнее варианта pluto (разница от 40% до 52% в зависимости от запуска). Результаты с OpenMP для программы floyd не совпадают при одинаковом программном коде из-за влияния других процессов операционной системы на вычисления. Результаты запуска floyd с MPI показывают преимущество ilpy перед pluto во всех запусках. Для программы gramschmidt в варианте OpenMP ilpy дал выигрыш в производитель-

ности в 5,37%. В запусках gramschmidt с MPI ilpy сравнивается с исходным последовательным вариантом программы, и преимущество от распараллеливания наблюдается только при запуске на одной машине и составляет 89%.

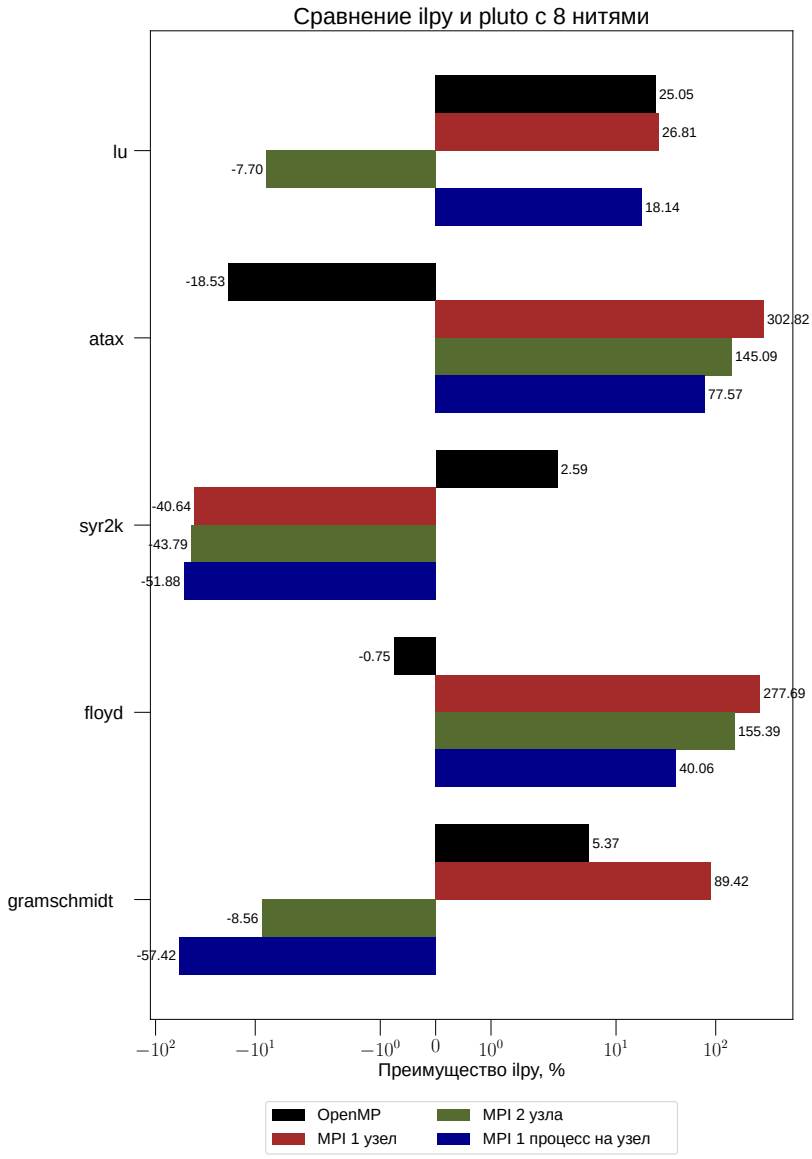


Рисунок 4.25 — Преимущество ilpy перед pluto при распараллеливании линейных программ в 8 нитях (процессах)

Можно заключить, что при распараллеливании с OpenMP ilpy показал преимущество для трех программ из пяти: для lu 25,05%, для syr2k 2,59%, для gramschmidt 5,37%. Идентичный результат был получен для программы floyd, а для программы atax был зафиксирован проигрыш на 18,53%. При распараллеливании с MPI ilpy показал преимущество для трех программ из пяти: для lu наблюдается преимущество в запусках на одной и восьми машинах, для atax и floyd наблюдается преимущество во всех вариантах запуска. Для

программы syr2k зафиксирован проигрыш, составляющий до 2 раз. Для программы gramschmidt нельзя сделать заключение, поскольку транслятор pluto не смог завершить работу. Результаты быстродействия параллельных программ, полученных применением ilpy, демонстрируют практическую применимость разработанного метода даже в условиях статического распределения нагрузки между MPI-процессами. В отдельных случаях использование топологической сортировки при нахождении первого компонента многомерного расписания вычислений, а также дублирование входных данных в вычислительных процессах может дополнительно улучшить производительность параллельной программы: для atax_p выигрыш ilpy составляет 2,62 раза с OpenMP и от 26,20 до 42,57 раз с MPI.

В результирующих параллельных программах количество аффинных отображений различалось в зависимости от применяемого транслятора. Для многомерных аффинных отображений, найденных ilpy и pluto, была выполнена оценка с помощью процедуры, представленной в приложении Ж, которая показала отсутствие явной линейной зависимости коэффициента ускорения от статистических характеристик величины (2.5). Разработка критериев оптимальности многомерного аффинного отображения является предметом дальнейших исследований.

Вопрос о времени, затрачиваемом на распараллеливание, также является важным в контексте работы программиста с кодом — разработка и отладка программ должны по возможности проходить в интерактивном режиме. Чрезмерно долгий процесс оптимизации и компиляции кода может этому помешать. Поэтому, чем быстрее выполняются оптимизирующие преобразования программ, тем эффективнее может быть работа программиста. Кроме того, встраивание оптимизирующих преобразований в среды с JIT-компиляцией особенно чувствительно к времени выполнения дополнительных оптимизаций, в особенности если они выполняются множество раз в циклических конструкциях, что может серьезно отразиться на итоговой производительности и нивелировать потенциальный выигрыш от применения параллельных вычислений. В таблице 5 приведено время работы транслятора ilpy в каждом эксперименте. Транслятор ilpy запускался на мобильном процессоре Intel(R) Core(TM) i5-11300H 3.10GHz. Время, затраченное транслятором, складывается из следующих составляющих: работа с файловой системой, работа библиотек clan, candl, cloog, вычисление весов зависимостей по данным и доступов к памяти, вычисление расписания вычислений, вычисление размещений вычислений и данных. Именно последний из указанных этапов в некоторых случаях оказывается чрезмерно ресурсоемким: поиск решения за-

дачи ЛЦП приходится ограничивать во времени. Этого удается достичь двумя различными способами. Во-первых, решателю glpk можно передать пороговое значение времени работы в секундах, по достижении которого работа будет остановлена (параметр `--iocp-tm-lim`). Если текущее найденное решение является допустимым, оно принимается как субоптимальное и используется в дальнейшей работе ilpy. Во-вторых, можно ограничить все переменные, участвующие в формулировке задачи ЛЦП, постоянным значением (верхняя граница задается параметром `--ilp-col-ub`). Каждый из этих способов позволяет быстрее найти субоптимальное решение, воздействуя на поиск решения задачи ЛЦП. Платой за ускорение распараллеливания служит отсутствие гарантии нахождения оптимума в том смысле, который отражен в (2.5) и (2.15).

Таблица 5 — Время работы транслятора ilpy

Строка запуска	Процедура	Трансляция, с	Вычисление расписания, с	Вычисление размещений, с
ilpy atax.c --cloog-f 1 --ilp-col-ub 2	`atax_ilp_sync`	281,760960	0,003236	281,657562
ilpy atax_p.c --arrays --smdp --out-fn-suffix arrays_mdp	`atax_p_ilp_arrays_mdp`	0,146954	0,000767	0,033877
ilpy atax_p.c --arrays --cloog-f 1	`atax_p_ilp_arrays`	0,186853	0,002203	0,030651
ilpy atax_p.c --smdp --out-fn-suffix deps_mdp	`atax_p_ilp_sync_mdp`	0,123167	0,000773	0,013306
ilpy atax_p.c --cloog-f 1	`atax_p_ilp_sync`	0,180710	0,002166	0,034623
ilpy floyd.c --arrays	`floyd_ilp_sync`	0,432201	0,003061	0,003684
ilpy floyd.c	`floyd_ilp_arrays`	0,702041	0,003041	0,281556
ilpy gramschmidt.c --arrays --cloog-f 1 --ilp-col-ub 2 --param-weight M=101,N=100	`gramschmidt_ilp_arrays`	113,671927	0,014190	110,040967
ilpy gramschmidt.c --cloog-f 1 --ilp-col-ub 2 --iocp-tm-lim 30 --param-weight M=101,N=100	`gramschmidt_ilp_sync`	33,712030	0,014484	30,008568
ilpy lu.c --arrays	`lu_ilp_arrays`	0,156310	0,003505	0,002766
ilpy lu.c	`lu_ilp_sync`	0,122583	0,002830	0,003117
ilpy syr2k.c --arrays --async --out-fn-suffix aarrays	`syr2k_ilp_arrays_async`	1,093218	0,001423	0,006617
ilpy syr2k.c --async	`syr2k_ilp_async`	1,095806	0,001390	0,028006
ilpy syr2k.c	`syr2k_ilp_sync`	1,084760	0,001420	0,001386

В трех случаях из четырнадцати рассмотренных время трансляции серьезно вышло за рамки одной секунды. Для процедур `atax_ilp_sync` и `gramschmidt_ilp_arrays` пришлось задать верхнюю границу для переменных, равную 2, но решатель завершил свою работу штатно, о чем свидетельствует сообщение `INTEGER OPTIMAL SOLUTION FOUND`. Для процедуры `gramschmidt_ilp_sync` пришлось дополнительно ограничить время работы решателя glpk тридцатью секундами (листинг Е.2). Решатель был завершен

принудительно (сообщение `TIME LIMIT EXCEEDED; SEARCH TERMINATED`), и текущее допустимое решение (предупреждение `[Warning] mip_status == GLP_FEAS`) было использовано как субоптимальное. Без применения этих способов время работы решателя glpk становилось неприемлемо долгим. Транслятор pluto не демонстрировал явных отклонений при выполнении экспериментов — во всех случаях, когда его работа завершалась успешно, затрачиваемое время не превышало пяти секунд.

Вопрос обеспечения стабильного времени работы компонента нахождения размещений вычислений и данных, а также применения разработанных компонентов для распараллеливания программ линейного класса в JIT-средах, остается открытым и также является предметом дальнейших исследований.

Заключение

Основной научный результат диссертации заключается в решении актуальной научной задачи, заключающейся в разработке методов нахождения пространственных и временных отображений программ линейного класса, обеспечивающих локальность использования данных при их параллельном выполнении на многопроцессорных вычислительных системах.

При проведении исследований и разработок по теме настоящей работы получены следующие результаты.

1. Установлены ограничения и недостатки существующих методов и средств нахождения пространственных и временных отображений программ линейного класса, ориентированных на распараллеливание гнезд циклов, обозначившие направления их совершенствования.
2. Разработаны новые критерии оптимальности пространственных и временных отображений программ линейного класса, отличающиеся возможностью ранжировать информационные зависимости и доступы к данным для более гибкого количественного описания локальности использования данных, чем целевые функции на основе лексикографического упорядочивания в методах П. Футриера и У. Бондхугулы.
3. Разработан новый метод нахождения оптимальных пространственных и временных отображений программ линейного класса для распараллеливания гнезд циклов, отличающийся применением взвешенной суммы показателей качества решения. Разработанный метод устраняет необходимость полного перебора решений с их трудоемкой оценкой качества (М. Грибль), и при этом позволяет следовать идее П. Футриера, но при ослабленных ограничениях, заключающихся в сокращении расстояния коммуникаций, а не полном их исключении, в меньшей степени ограничивая параллелизм.
4. Разработан метод генерации параллельной MPI-программы, позволяющий организовать информационный обмен между параллельными процессами в случае явно заданного распределения данных между процессорами. По сравнению с методом Р. Дататри, исключающим дублирование информации при пересылке информационных пакетов, нет

необходимости размещать входные данные на всех вычислительных устройствах.

5. Получены экспериментальные результаты исследования производительности параллельных вариантов программ (lu, atax, syr2k, floyd, gramschmidt), свидетельствующие о выигрыше в эффективности распараллеливания по сравнению с современным транслятором Pluto: до 25% (с OpenMP) и до 302% (с MPI). Разработаны компоненты, которые могут быть использованы в автоматически распараллеливающем трансляторе для поддержки распараллеливания программ, написанных на языке Си.

Разработанные при решении научной задачи методы и средства позволили достигнуть поставленной цели, заключающейся в повышении быстродействия программ, получаемых в результате применения средств автоматического распараллеливания. Результаты диссертационного исследования подтверждены экспериментальными исследованиями и использованы в ООО «НПП САТЭК плюс», г. Рыбинск, и в учебном процессе кафедр КБ-4 и КБ-14 Института кибербезопасности и цифровых технологий Федерального государственного бюджетного образовательного учреждения высшего образования «МИРЭА — Российский технологический университет».

Словарь терминов

ПО : Программное обеспечение.

ЛП : Линейное программирование.

ЛЦП : Линейное целочисленное программирование.

JIT : Just-In-Time (трансляция) — компиляции байт-кода в машинный код или в другой формат непосредственно во время работы программы.

GNU Make : Утилита, автоматизирующая процесс преобразования файлов из одной формы в другую.

gcc : Компилятор языка Си.

MPI : Message Passing Interface — стандарт передачи сообщений для кластеров и суперкомпьютеров.

GPU : Graphics Processing Unit — графический ускоритель.

CUDA : Compute Unified Device Architecture — программно-аппаратная архитектура параллельных вычислений, разработанная для выпускаемых NVIDIA графических ускорителей.

SIMD : Single Instruction Multiple Data (одиночный поток команд, множественный поток данных) — принцип компьютерных вычислений, позволяющий обеспечить параллелизм на уровне данных.

Список литературы

1. *Воеводин, В. В.* Параллельные вычисления [Текст] / В. В. Воеводин, В. В. В. — БХВ-Петербург, 2002. — С. 608.
2. *Feautrier, P.* Polyhedron Model. [Текст] / P. Feautrier, C. Lengauer // Encyclopedia of parallel computing. — 2011. — Т. 1. — С. 1581—1592.
3. Компиляторы: принципы, технологии и инструментарий [Текст] / А. Ахо [и др.]. — 2008.
4. *Ларкин, Е. В.* Прогнозирование времени выполнения алгоритма [Текст] / Е. В. Ларкин, А. Н. Ивутин // Известия Тульского государственного университета. Технические науки. — 2013. — № 3. — С. 301—315.
5. *Griebl, M.* Automatic parallelization of loop programs for distributed memory architectures [Текст] / M. Griebl. — Univ. Passau, Passau, Germany, 2004. — С. 207.
6. *Якобовский, М. В.* Введение в параллельные методы решения задач [Текст] / М. В. Якобовский ; предисл. В. А. Садовничий. — Издательство Московского университета, 2013. — С. 328. — (Суперкомпьютерное образование).
7. *Kumar, K. G.* Generalized Unimodular Loop Transformations for Distributed Memory Multiprocessors. [Текст] / K. G. Kumar, D. Kulkarni, A. Basu // ICPP (2). — Citeseer. 1991. — С. 146—149.
8. *Kumar, K. G.* Deriving good transformations for mapping nested loops on hierarchical parallel machines in polynomial time [Текст] / K. G. Kumar, D. Kulkarni, A. Basu // Proceedings of the 6th international conference on Supercomputing. — 1992. — С. 82—92.
9. *Kelly, W.* A unifying framework for iteration reordering transformations [Текст] / W. Kelly, W. Pugh // Proceedings 1st International Conference on Algorithms and Architectures for Parallel Processing. Т. 1. — IEEE. 1995. — С. 153—162.
10. *Wolf, M. E.* A loop transformation theory and an algorithm to maximize parallelism [Текст] / M. E. Wolf, M. S. Lam // IEEE Transactions on Parallel & Distributed Systems. — 1991. — Т. 2, № 04. — С. 452—471.
11. *Banerjee, U.* Loop transformations for restructuring compilers: the foundations [Текст] / U. Banerjee. — Springer Science & Business Media, 2007. — С. 305.

12. *Wolfe, M. J.* High performance compilers for parallel computing [Текст] / M. J. Wolfe. — Addison-Wesley Longman Publishing Co., Inc., 1996. — С. 570.

13. Открытая распараллеливающая система 2006 [Текст] / Б. Я. Штейнберг [и др.] // Труды III международной конференции Параллельные вычисления и задачи управления, PACO. — 2006. — С. 526—541.

14. *Штейнберг, Б. Я.* Математические методы распараллеливания рекуррентных циклов для суперкомпьютеров с параллельной памятью [Текст] / Б. Я. Штейнберг. — 2004.

15. Автоматизация распараллеливания программ с блочным размещением данных [Текст] / Л. Р. Гервич [и др.] // Сибирский журнал вычислительной математики. — 2015. — Т. 18, № 1. — С. 41—53.

16. *Клинов, М. С.* Автоматическое распараллеливание Фортран-программ. Отображение на кластер [Текст] / М. С. Клинов, В. А. Крюков // Вестник Нижегородского университета им. Н. И. Лобачевского. — 2009. — Т. 2. — С. 128—134.

17. Автоматическое распараллеливание Фортран-программ на кластер с графическими ускорителями [Текст] / В. А. Бахтин [и др.] // Сборник трудов Международной научной конференции «Параллельные вычислительные технологии» (ПаВТ’2012), Новосибирск. — 2012. — С. 373—379.

18. Новые возможности DVM-системы [Текст] / В. Ф. Алексахин [и др.] // Научный сервис в сети Интернет. Т. 21. — Федеральное государственное учреждение «Федеральный исследовательский центр Институт прикладной математики им. М. В. Келдыша Российской академии наук». 2019. — С. 25—39.

19. Модульная архитектура компилятора языка Норма+ [Текст] / А. Н. Андрианов [и др.] // Препринты Института прикладной математики им. М. В. Келдыша РАН. — 2011. — Т. 64. — С. 1—16.

20. Язык Норма [Текст] / А. Н. Андрианов [и др.] // Препринты Института прикладной математики им. М. В. Келдыша РАН. — 2019. — Т. 132. — С. 1—48.

21. Decoupling algorithms from schedules for easy optimization of image processing pipelines [Текст] / J. Ragan-Kelley [и др.] // ACM Transactions on Graphics (TOG). — 2012. — Т. 31, № 4. — С. 1—12.

22. *Lengauer, C.* Loop parallelization in the polytope model [Текст] / C. Lengauer // International Conference on Concurrency Theory. — Springer. 1993. — С. 398—416.

23. *Größlinger, A.* The challenges of non-linear parameters and variables in automatic loop parallelisation [Текст] / A. Größlinger. — Lulu. com, 2010.

24. *Größlinger, A.* Introducing non-linear parameters to the polyhedron model [Текст] / A. Größlinger, M. Griebl, C. Lengauer // Proc. 11th Workshop on Compilers for Parallel Computers (CPC 2004), Research Report Series. — Citeseer. 2004. — С. 1—12.

25. *Barthou, D.* Array dataflow analysis in presence of non-affine constraints [Текст] / D. Barthou. — 1998. — С. 208.

26. FADAlib: an open source C++ library for fuzzy array dataflow analysis [Текст] / M. Belaoucha, D. Barthou, A. Eliche [и др.] // Procedia Computer Science. — 2010. — Т. 1, № 1. — С. 2075—2084.

27. *Feautrier, P.* Some efficient solutions to the affine scheduling problem. I. One-dimensional time [Текст] / P. Feautrier // International journal of parallel programming. — 1992. — Т. 21. — С. 313—347.

28. *Ортега, Д.* Введение в параллельные и векторные методы решения линейных систем [Текст] / Д. Ортега. — Мир, 1991. — С. 367.

29. *Feautrier, P.* Dataflow analysis of array and scalar references [Текст] / P. Feautrier // International Journal of Parallel Programming. — 1991. — Т. 20. — С. 23—53.

30. *Арапбаев, Р. Н.* Сравнительный анализ тестов на зависимость по данным [Текст] / Р. Н. Арапбаев, В. А. Евстигнеев, Р. А. Осмонов // Новосибирск: Институт систем информатики им. А. П. Ершова СО РАН. — 2006. — С. 37.

31. *Petersen, P. M.* Static and dynamic evaluation of data dependence analysis techniques [Текст] / P. M. Petersen, D. A. Padua // IEEE Transactions on Parallel and Distributed Systems. — 1996. — Т. 7, № 11. — С. 1121—1132.

32. *Wolfe, M. J.* Triangular Banerjee's Inequalities with Directions [Текст] / M. J. Wolfe. — Oregon Graduate Institute of Science, Technology, Department of Computer Science, Engineering, 1992.

33. *Feautrier, P.* Parametric integer programming [Текст] / P. Feautrier // RAIRO-Operations Research. — 1988. — Т. 22, № 3. — С. 243—268.

34. *Collard, J.-F.* A precise fixpoint reaching definition analysis for arrays [Текст] / J.-F. Collard, M. Griebl // Languages and Compilers for Parallel Computing: 12th International Workshop, LCPC'99 La Jolla, CA, USA, August 4–6, 1999 Proceedings 12. — Springer. 2000. — С. 286—302.

35. *Griebl, M.* The loop parallelizer LooPo—announcement [Текст] / M. Griebl, C. Lengauer // International Workshop on Languages and Compilers for Parallel Computing. — Springer. 1996. — С. 603—604.

36. *Bastoul, C.* Candl: the chunky analyzer for dependences in loops [Текст] : тех. отч. / C. Bastoul, L.-N. Pouchet ; tech. rep., LRI, Paris-Sud University, France. — 2012.

37. Automatic transformations for communication-minimized parallelization and locality optimization in the polyhedral model [Текст] / U. Bondhugula [и др.] // Compiler Construction: 17th International Conference, CC 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings 17. — Springer. 2008. — С. 132—146.

38. A practical automatic polyhedral parallelizer and locality optimizer [Текст] / U. Bondhugula [и др.] // Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation. — 2008. — С. 101—113.

39. *Лебедев, А. С.* Система автоматического распараллеливания линейных программ для машин с общей и распределенной памятью [Электронный ресурс] / А. С. Лебедев, Ш. Г. Магомедов // Российский технологический журнал. — 2019. — Т. 7, № 5. — С. 7—19. — URL: https://www.rtj-mirea.ru/jour/article/view/168.

40. *Лебедев, А. С.* Трансляция программ линейного класса для параллельного выполнения на универсальных многоядерных процессорах [Текст] / А. С. Лебедев, В. И. Солодовников // Труды Института системного анализа Российской академии наук. — 2023. — Т. 73, № 4. — С. 36—47.

41. *Лебедев, А. С.* Трансляция программ линейного класса для параллельного выполнения на системах с распределенной памятью [Текст] / А. С. Лебедев // Системы высокой доступности. — 2023. — Т. 19, № 3. — С. 35—47.

42. *Свидетельство о гос. регистрации программы для ЭВМ*. Модуль вычисления пространственно-временного преобразования линейной программы с целью ее распараллеливания и оптимизации локальности данных [Текст] : 2016618301 Рос. Федерация / А. С. Лебедев ; О. с ограниченной ответственностью Технологии высокопроизводительных вычислений. — № 2016612892 ; заявл. 30.03.2016 ; опубл. 20.08.2016.

43. *Свидетельство о гос. регистрации программы для ЭВМ*. Программа для построения аффинных преобразований программ линейного класса [Текст] : 2022667001 Рос. Федерация / А. С. Лебедев, С. В. И. ; Ф. государственное бюджетное учреждение науки Центр информационных технологий в проектировании Российской академии наук. — № 2022666032 ; заявл. 29.08.2022 ; опубл. 13.09.2022.

44. *Lamport, L.* The parallel execution of DO loops [Текст] / L. Lamport // Communications of the ACM. — 1974. — Т. 17, № 2. — С. 83—93.

45. *Feautrier, P.* Some efficient solutions to the affine scheduling problem. Part II. Multidimensional time [Текст] / P. Feautrier // International journal of parallel programming. — 1992. — Т. 21. — С. 389—420.

46. Tutorial PIPS: An Interprocedural, Extensible, Source-to-Source Compiler Infrastructure for Code Transformations and Instrumentations [Текст] / C. Ancourt [и др.] // CGO 2011: International Symposium on Code Generation and Optimization. — 2011.

47. *Konstantinidis, A. A.* More definite results from the Pluto scheduling algorithm [Текст] / A. A. Konstantinidis, P. H. Kelly // 1st International Workshop on Polyhedral Compilation Techniques (IMPACT), C. Alias and C. Bastoul (Eds.). Chamonix, France. — 2011.

48. Affine transformations for communication minimal parallelization and locality optimization of arbitrarily nested loop sequences [Текст] / U. Bondhugula [и др.] // Technical Report. — 2007.

49. *Verdoolaege, S.* isl: An integer set library for the polyhedral model [Текст] / S. Verdoolaege // International Congress on Mathematical Software. — Springer. 2010. — С. 299—302.

50. *Grosser, T. C.* Enabling polyhedral optimizations in llvm [Текст] / T. C. Grosser. — Univ. Passau, Passau, Germany, 2011. — С. 101.

51. *Grosser, T.* Polly — performing polyhedral optimizations on a low-level intermediate representation [Текст] / T. Grosser, A. Groesslinger, C. Lengauer // Parallel Processing Letters. — 2012. — Т. 22, № 04. — С. 1250010.

52. Design of graphite and the polyhedral compilation package [Текст] / J. Sjödin [и др.] // GCC Developers' Summit. — 2009.

53. *Лебедев, А. С.* Оптимизация временной локальности данных при автоматическом распараллеливании линейных программ [Электронный ресурс] / А. С. Лебедев // Современные проблемы науки и образования. — 2014. — № 6. — С. 192—192. — URL: https://science-education.ru/ru/article/view?id=16255.

54. *Лебедев, А. С.* Пространственно-временные преобразования при распараллеливании линейных программ [Текст] / А. С. Лебедев // Информационные технологии и вычислительные системы. — 2015. — № 1. — С. 19—32.

55. *Лебедев, А. С.* Оптимизация локальности данных при автоматическом распараллеливании программ [Электронный ресурс] / А. С. Лебедев // Сборник тезисов докладов Третьего Национального суперкомпьютерного форума (НСКФ-2014) (Переславль-Залесский 25-27 ноября 2014 г.). — Переславль-Залесский: Институт программных систем имени А. К. Айламазяна Российской академии наук, 2014. — 2014. — URL: https://2014.nscf.ru/TesisAll/4_Systemnoe_i_promezhytochnoe_PO/11_179_LebedevAS.pdf.

56. *Лебедев, А. С.* Среда автоматического распараллеливания программ для систем с общей и распределенной памятью [Электронный ресурс] / А. С. Лебедев // Сборник тезисов докладов Пятого Национального суперкомпьютерного форума (НСКФ-2016) (Переславль-Залесский, 29 ноября — 02 декабря 2016 г.). — Переславль-Залесский: Институт программных систем имени А. К. Айламазяна Российской академии наук, 2016. — 2016. — URL: https://2016.nscf.ru/TesisAll/03_Systemnoe_i_promezhytochnoe_PO/732_LlebedevAS.pdf.

57. *Lebedev, A. S.* Construction of optimal space-time mappings for automatic parallelization of loop nests with static control flow [Text] / A. S. Lebedev // 2017 IEEE 11th International Conference on Application of Information and Communication Technologies (AICT). — IEEE. 2017. — P. 1—7.

58. *Лебедев, А. С.* Построение пространственных и временных преобразований программ для распараллеливания вложенностей циклов [Текст] / А. С. Лебедев // Многопроцессорные вычислительные и управляющие системы : материалы Всероссийской научно-технической конференции (МВУС-2022), Таганрог, 27—30 июня 2022 г. — Ростов-на-Дону: Южный федеральный университет, 2022. — Издательство Южного федерального университета, 2022. — С. 90—95.

59. Hybrid static/dynamic schedules for tiled polyhedral programs [Текст] / T. Jin [и др.] // arXiv preprint arXiv:1610.07236. — 2016.

60. Dynamic and speculative polyhedral parallelization using compiler-generated skeletons [Текст] / A. Jimborean [и др.] // International Journal of Parallel Programming. — 2014. — Т. 42. — С. 529—545.

61. *Pradelle, B.* Static and dynamic methods of polyhedral compilation for an efficient execution in multicore environments [Текст] / B. Pradelle. — 2011.

62. *Sukumaran Rajam, A.* Beyond the realm of the polyhedral model: combining speculative program parallelization with polyhedral compilation [Текст] / A. Sukumaran Rajam. — 2015.

63. *Lim, A. W.* Maximizing parallelism and minimizing synchronization with affine transforms [Текст] / A. W. Lim, M. S. Lam // Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. — 1997. — С. 201—214.

64. *Lim, A. W.* Maximizing parallelism and minimizing synchronization with affine partitions [Текст] / A. W. Lim, M. S. Lam // Parallel computing. — 1998. — Т. 24, № 3/4. — С. 445—475.

65. *Lim, A. W.* An affine partitioning algorithm to maximize parallelism and minimize communication [Текст] / A. W. Lim, G. I. Cheong, M. S. Lam // Proceedings of the 13th international conference on Supercomputing. — 1999. — С. 228—237.

66. *Feautrier, P.* Automatic distribution of data and computations [Текст] / P. Feautrier // TSI-Technique et Science Informatiques-RAIRO. — 1996. — Т. 15, № 5. — С. 529—558.

67. *Feautrier, P.* Toward automatic distribution [Текст] / P. Feautrier // Parallel Processing Letters. — 1994. — Т. 4, № 03. — С. 233—244.

68. *Лебедев, А. С.* Размещение данных при автоматическом распараллеливании линейных программ для систем с распределенной памятью [Текст] / А. С. Лебедев // Вестник Рыбинской государственной авиационной технологической академии им. П. А. Соловьева. — 2015. — № 3. — С. 92—99.

69. *Lebedev, A. S.* Automatic Parallelization of Affine Programs for Distributed Memory Systems [Text] / A. S. Lebedev, S. G. Magomedov // Futuristic Trends in Network and Communication Technologies: Third International Conference, FTNCT 2020, Taganrog, Russia, October 14–16, 2020, Revised Selected Papers, Part II 3. — Springer. 2021. — P. 91—101.

70. *Lam, M. S.* A data locality optimizing algorithm [Текст] / M. S. Lam, M. E. Wolf // ACM SIGPLAN Notices. — 2004. — Т. 39, № 4. — С. 442—459.

71. *Edin, H.* On Supernode Partitioning Hyperplanes for Two Dimensional Algorithms [Текст] / H. Edin, W. Shang. — 1997.

72. *Perepelkina, A. Y.* The DiamondCandy algorithm for maximum performance vectorized cross-stencil computation [Текст] / A. Y. Perepelkina, V. D. Levchenko // Препринты Института прикладной математики им. М. В. Келдыша РАН. — 2018. — Т. 225. — С. 1—23.

73. *Perepelkina, A. Y.* Diamond Torre Algorithm for High-Performance Wave Modeling [Текст] / A. Y. Perepelkina, V. D. Levchenko // Препринты Института прикладной математики им. М. В. Келдыша РАН. — 2015. — Т. 18. — С. 1—20.

74. *Bandishti, V.* Tiling stencil computations to maximize parallelism [Текст] / V. Bandishti, I. Pananilath, U. Bondhugula // SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. — IEEE. 2012. — С. 1—11.

75. Tiling and optimizing time-iterated computations on periodic domains [Текст] / U. Bondhugula [и др.] // Proceedings of the 23rd international conference on Parallel architectures and compilation. — 2014. — С. 39—50.

76. *Левченко, А. В.* Метод гибридного неоднородного тайлинга для архитектур суперкомпьютеров с многоуровневой иерархией памяти [Текст] / А. В. Левченко // Информатика, телекоммуникации и управление. — 2019. — Т. 12, № 4. — С. 29—44.

77. *Reddy, C.* Effective automatic computation placement and data allocation for parallelization of regular programs [Текст] / C. Reddy, U. Bondhugula // Proceedings of the 28th ACM international conference on Supercomputing. — 2014. — С. 13—22.

78. *Lebedev, A.* On tiling for heterogeneous systems during mechanical code parallelization [Text] / A. Lebedev // Теория и практика системного анализа: Труды III Всероссийской научной конференции молодых ученых с международным участием. – ТІ – Рыбинск: РГАТУ имени П. А. Соловьева, 2014. – 200 с. — 2014. — P. 151—156.

79. *Ramashekar, T.* Automatic data allocation and buffer management for multi-GPU machines [Текст] / T. Ramashekar, U. Bondhugula // ACM Transactions on Architecture and Code Optimization (TACO). — 2013. — Т. 10, № 4. — С. 1—26.

80. *Baskaran, M. M.* Automatic C-to-CUDA code generation for affine programs [Текст] / M. M. Baskaran, J. Ramanujam, P. Sadayappan // Compiler Construction: 19th International Conference, CC 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings 19. — Springer. 2010. — С. 244—263.

81. Polyhedral parallel code generation for CUDA [Текст] / S. Verdoolaege [и др.] // ACM Transactions on Architecture and Code Optimization (TACO). — 2013. — Т. 9, № 4. — С. 1—23.

82. Automatic data movement and computation mapping for multi-level parallel architectures with explicitly managed memories [Текст] / M. M. Baskaran [и др.] // Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming. — 2008. — С. 1—10.

83. A compiler framework for optimization of affine loop nests for GPGPUs [Текст] / M. M. Baskaran [и др.] // Proceedings of the 22nd annual international conference on Supercomputing. — 2008. — С. 225—234.

84. *Größlinger, A.* Precise management of scratchpad memories for localising array accesses in scientific codes [Текст] / A. Größlinger // Compiler Construction: 18th International Conference, CC 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings 18. — Springer. 2009. — С. 236—250.

85. *Baghdadi, S.* Putting automatic polyhedral compilation for GPGPU to work [Текст] / S. Baghdadi, A. Größlinger, A. Cohen // Proceedings of the 15th Workshop on Compilers for Parallel Computers (CPC'10). — 2010.

86. *Katel, N.* MLIR-based code generation for GPU tensor cores [Текст] / N. Katel, V. Khandelwal, U. Bondhugula // Proceedings of the 31st ACM SIGPLAN International Conference on Compiler Construction. — 2022. — С. 117—128.

87. When polyhedral transformations meet SIMD code generation [Текст] / M. Kong [и др.] // Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation. — 2013. — С. 127—138.

88. *Bastoul, C.* Code generation in the polyhedral model is easier than you think [Текст] / C. Bastoul // Proceedings. 13th International Conference on Parallel Architecture and Compilation Techniques, 2004. PACT 2004. — IEEE. 2004. — С. 7—16.

89. *Bondhugula, U.* Compiling affine loop nests for distributed-memory parallel architectures [Текст] / U. Bondhugula // Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. — 2013. — С. 1—12.

90. Generating efficient data movement code for heterogeneous architectures with distributed-memory [Текст] / R. Dathathri [и др.] // Proceedings of the 22nd international conference on Parallel architectures and compilation techniques. — IEEE. 2013. — С. 375—386.

91. *Лебедев, А. С.* Организация информационного обмена между параллельными процессами при автоматическом распараллеливании линейных программ для кластерных систем с применением модели многогранников [Электронный ресурс] / А. С. Лебедев // Программные системы: теория и приложения. — 2017. — Т. 8, 4 (35). — С. 3—20. — URL: https://psta.psiras.ru/read/psta2017_4_3-20.pdf.

92. *Saà-Garriga, A.* OMP2MPI: Automatic MPI code generation from OpenMP programs [Текст] / A. Saà-Garriga, D. Castells-Rufas, J. Carrabina // arXiv preprint arXiv:1502.02921. — 2015.

93. From OpenMP to MPI: first experiments of the STEP source-to-source transformation tool [Текст] / D. Millot [и др.] // ParCO 2009: International Conference on Parallel Computing: Mini-Symposium «Parallel Programming Tools for Multi-core Architectures». — IOS Press. 2009. — С. 669—676.

94. STEP: a distributed OpenMP for coarse-grain parallelism tool [Текст] / D. Millot [и др.] // Lecture Notes in Computer Science. — 2008. — Т. 5004. — С. 83—99.

95. *Morvan, A.* Efficient nested loop pipelining in high level synthesis using polyhedral bubble insertion [Текст] / A. Morvan, S. Derrien, P. Quinton // 2011 International Conference on Field-Programmable Technology. — IEEE. 2011. — С. 1—10.

96. OP2: An active library framework for solving unstructured mesh-based applications on multi-core and many-core architectures [Текст] / G. R. Mudalige [и др.] // 2012 Innovative Parallel Computing (InPar). — 2012. — С. 1—12.

97. A domain-specific approach to heterogeneous parallelism [Текст] / H. Chafi [и др.] // ACM SIGPLAN Notices. — 2011. — Т. 46, № 8. — С. 35—46.

98. OptiML: an implicitly parallel domain-specific language for machine learning [Текст] / A. Sujeeth [и др.] // Proceedings of the 28th International Conference on Machine Learning (ICML-11). — 2011. — С. 609—616.

99. PENCIL: Towards a platform-neutral compute intermediate language for DSLs [Текст] / R. Baghdadi [и др.] // arXiv preprint arXiv:1302.5586. — 2013.

100. Pencil: A platform-neutral compute intermediate language for accelerator programming [Текст] / R. Baghdadi [и др.] // 2015 International Conference on Parallel Architecture and Compilation (PACT). — IEEE. 2015. — С. 138—149.

101. Polyhedral optimizations for a data-flow graph language [Текст] / A. Sbîrlea [и др.] // International Workshop on Languages and Compilers for Parallel Computing. — Springer. 2015. — С. 57—72.

102. Predictive modeling in a polyhedral optimization space [Текст] / E. Park [и др.] // International journal of parallel programming. — 2013. — Т. 41, № 5. — С. 704—750.

103. *Clauss, P.* Deriving formulae to count solutions to parameterized linear systems using Ehrhart polynomials: Applications to the analysis of nested-loop programs [Текст] / P. Clauss, V. Loechner, D. Wilde // ICPS RR. — 1997. — С. 97—05.

104. *Mitchell, J. E.* Branch-and-cut algorithms for combinatorial optimization problems [Текст] / J. E. Mitchell // Handbook of applied optimization. — 2002. — Т. 1, № 1. — С. 65—77.

105. *Griebl, M.* Forward communication only placements and their use for parallel program construction [Текст] / M. Griebl, P. Feautrier, A. Größlinger // Languages and Compilers for Parallel Computing: 15th Workshop, LCPC 2002, College Park, MD, USA, July 25-27, 2002. Revised Papers 15. — Springer. 2005. — С. 16—30.

106. *Черникова, Н. В.* Алгоритм для нахождения общей формулы неотрицательных решений системы линейных неравенств [Текст] / Н. В. Черникова // Журнал вычислительной математики и математической физики. — 1965. — Т. 5, № 2. — С. 334—337.

107. *Le Verge, H.* A note on Chernikova's algorithm [Текст] / H. Le Verge. — 1992.

108. *Loechner, V.* PolyLib: A library for manipulating parameterized polyhedra [Текст] / V. Loechner. — 1999.

109. *Bondhugula, U.* Handling Negative Coefficients in Automatic Transformation Schedules [Текст] / U. Bondhugula, A. Cohen. — 2014.

110. *Pouchet, L.-N.* PolyBench/C 4.1 [Электронный ресурс] / L.-N. Pouchet, T. Yuki. — 2015. — URL: https://sourceforge.net/projects/polybench/.

111. *Bastoul, C.* Openscop: A specification and a library for data exchange in polyhedral compilation tools [Текст] / C. Bastoul // Paris-Sud University, France, Tech. Rep. — 2011. — Т. 9. — С. 22.

112. *Godbolt, M.* Optimizations in C++ Compilers: A practical journey [Текст] / M. Godbolt // Queue. — 2019. — Т. 17, № 5. — С. 69—100.

113. *Bastoul, C.* Extracting polyhedral representation from high level languages [Текст] / C. Bastoul // Tech. rep. Related to the Clan tool. LRI, Paris-Sud University. — 2008.

114. *Евстигнеев, В.* Многочлены Эрхарта [Текст] / В. Евстигнеев // Программные средства и математические основы информатики. — 2004. — С. 60—79.

115. *Makhorin, A.* GLPK (GNU linear programming kit) [Электронный ресурс] / A. Makhorin. — 2014. — URL: https://www.gnu.org/software/glpk/.

116. *Xianyi, Z.* OpenBLAS: An optimized BLAS library [Текст] / Z. Xianyi, W. Qian, W. Saar. — 2023. — URL: https://www.openblas.net/.

117. *Лиходед, Н.* Методы распараллеливания гнезд циклов [Текст] / Н. Лиходед. — 2008.

118. *Белоусов, А.* Дискретная математика [Текст] / А. Белоусов, С. Ткачев. — 2015.

119. *Graham-Cumming, J.* The GNU make book [Текст] / J. Graham-Cumming. — No Starch Press, 2015.

120. *Gough, B. J.* An Introduction to GCC. [Текст] / B. J. Gough, R. Stallman. — Network Theory Limited, 2004.

121. *Marsaglia, G.* The structure of linear congruential sequences [Текст] / G. Marsaglia // Applications of number theory to numerical analysis. — Elsevier, 1972. — С. 249—285.

122. *Graham, R. L.* Open MPI: A flexible high performance MPI [Текст] / R. L. Graham, T. S. Woodall, J. M. Squyres // Parallel Processing and Applied Mathematics: 6th International Conference, PPAM 2005, Poznań, Poland, September 11-14, 2005, Revised Selected Papers 6. — Springer. 2006. — С. 228—239.

123. *Yoo, A. B.* Slurm: Simple linux utility for resource management [Текст] / A. B. Yoo, M. A. Jette, M. Grondona // Workshop on job scheduling strategies for parallel processing. — Springer. 2003. — С. 44—60.

124. *Leon, S. J.* Gram-Schmidt orthogonalization: 100 years and more [Текст] / S. J. Leon, Å. Björck, W. Gander // Numerical Linear Algebra with Applications. — 2013. — Т. 20, № 3. — С. 492—532.

Список рисунков

Список таблиц

Приложение А

Утилиты для поддержки тестовых запусков программ

А.1 Библиотека макросов информационного обмена blockdist.h

Разработанная на языке С библиотека макросов blockdist.h предоставляет средства для обработки многогранников коммуникаций в параллельных программах, предполагающих двухстороннюю коммуникацию процессов в рамках стандарта MPI. Разработаны макросы для работы с информационными пакетами, оборачивающими строки и столбцы матриц, линеаризованных по строкам и распределенных в вычислительных процессах согласно блочной схеме распределения.

Разработанные макросы могут быть включены в параллельную программу для реализации операций удаленного чтения и удаленной записи согласно схемам на листингах 3.1 и 3.2.

А.1.1 Вспомогательные конструкции и распределение массивов

Глобальные переменные `Q`, `R`, `L`, `U` представляют одноименные величины, введенные в определениях (3.1) и используемые в схемах 3.1, 3.2. Их значения должны быть заданы извне перед началом работы. Значения `Q` и `R` могут быть инициализированы непосредственными вызовами функций MPI:

```
MPI_Comm_size(MPI_COMM_WORLD, &Q);
MPI_Comm_rank(MPI_COMM_WORLD, &R);
```

Значения `L` и `U` задаются программистом в согласии с определениями (3.1).

Сопоставление виртуальных и физических процессоров влечет распределение вычислений и данных по физическим процессорам в соответствии с найденными размещениями вычислений и данных для каждой инструкции и массива в программе. Разработан набор макросов, упрощающий это сопоставление (листинг А.1): с помощью `blockdist_chunk_size` можно распределить `N` объек-

тов по `Q` корзинам, получив размер порции (в дальнейшем он будет называться `chunk_size`) во всех корзинах, кроме последней занятой, в которой может оказаться меньшее число объектов. Действительное число объектов в корзине с индексом `R` дает `blockdist_actual_chunk_size_r` (в дальнейшем оно будет называться `actual_chunk_size`). Корзины можно конкретизировать до физических процессоров, а в качестве объектов могут выступать элементы массивов, а также виртуальные процессоры и сопоставляемые им вычисления.

Листинг А.1 Размеры порций объектов при раскладке по корзинам

```
1 #define blockdist_chunk_size(N, Q) (((N) + (Q) - 1) / (Q))
2 #define blockdist_actual_chunk_size_r(chunk_size, N, R) max(0, min((chunk_size), (N) - (R) *
    ↪ (chunk_size)))
3 #define blockdist_actual_chunk_size(chunk_size, N) blockdist_actual_chunk_size_r(chunk_size, N, R)
```

Начальные и конечные индексы объектов, попавших в корзины, можно определить применением `blockdist_chunk_start_r` и `blockdist_chunk_end_r` (листинг А.2). Все макросы с суффиксом `_r` в названии требуют указания номера корзины, а без него — используют в качестве номера корзины значение глобальной переменной `R`, ассоциируя корзину с физическим процессором, на котором выполняется код.

Листинг А.2 Начальные и конечные индексы объектов в корзинах

```
1 #define blockdist_chunk_start_r(chunk_size, R) ((R) * (chunk_size))
2 #define blockdist_chunk_start(chunk_size) blockdist_chunk_start_r(chunk_size, R)
3 #define blockdist_chunk_end_r(chunk_size, actual_chunk_size, R) (blockdist_chunk_start_r(chunk_size,
    ↪ R) + (actual_chunk_size))
4 #define blockdist_chunk_end(chunk_size, actual_chunk_size) blockdist_chunk_end_r(chunk_size,
    ↪ actual_chunk_size, R)
```

Инициализация библиотеки происходит в вызове `blockdist_init` перед началом работы (листинг А.3). При этом инициализируются глобальные переменные `block_size` и `actual_block_size`, хранящие размер порции виртуальных процессоров, сопоставленной всем физическим процессорам, кроме последнего занятого, и действительное количество виртуальных процессоров, сопоставленных данному физическому соответственно. Глобальные переменные `lR` и `uR` будут хранить значения $l(R)$ и $u(R)$ в согласии с определениями (3.1) для физического процессора, на котором исполняется код (процессор `R`). Проверить, сопоставлен ли виртуальный процессор `v` данному физическому процессору,

можно с помощью `blockdist_in_range`. Узнать номер физического процессора, которому сопоставлен виртуальный процессор `v`, можно применением `blockdist_proc_rank`. Распределение данных выполняется аналогично распределению вычислений в `blockdist_array_distribute_r`, если найденное размещение данных имеет вид $\eta_A(\vec{i}, \vec{z}) = \vec{i}^{(k)}, k = 0, \ldots, p_A - 1$ для рассматриваемого массива `A`.

Листинг А.3 Инициализация библиотеки и распределение массивов

```
1  #define blockdist_nb_vproc (U - L + 1)
2  void blockdist_init() {
3    block_size = blockdist_chunk_size(blockdist_nb_vproc, Q);
4    actual_block_size = blockdist_actual_chunk_size(block_size, blockdist_nb_vproc);
5    lR = L + R * block_size;
6    uR = lR + actual_block_size - 1;
7  }
8  #define blockdist_array_distribute_r(N, R) blockdist_actual_chunk_size_r(block_size, N, R)
9  #define blockdist_array_distribute(N) blockdist_array_distribute_r(N, R)
10 #define blockdist_in_range(v) ((v) >= lR && (v) <= uR)
11 #define blockdist_proc_rank(v) (((v) - L) / block_size)
```

Если раскладка объектов по корзинам должна производиться в обратном порядке (например, при найденном размещении данных в виде $\eta_A(\vec{i}, \vec{z}) = \vec{z}^{(k)} - \vec{i}^{(k)}, k = 0, \ldots, p_A - 1$ для рассматриваемого массива `A`), то начальные и конечные индексы объектов, попавших в корзины, можно определить применением `blockdist_chunk_start_inv_r` и `blockdist_chunk_end_inv_r` (листинг А.4).

Листинг А.4 Начальные и конечные индексы объектов в корзинах при раскладке в обратном порядке

```
1  #define blockdist_chunk_start_inv_r(N, chunk_size, actual_chunk_size, R) ((N) -
       ↪ blockdist_chunk_end_r(chunk_size, actual_chunk_size, R))
2  #define blockdist_chunk_start_inv(N, chunk_size, actual_chunk_size) blockdist_chunk_start_inv_r(N,
       ↪ chunk_size, actual_chunk_size, R)
3  #define blockdist_chunk_end_inv_r(N, chunk_size, actual_chunk_size, R)
       ↪ (blockdist_chunk_start_inv_r(N, chunk_size, actual_chunk_size, R) + (actual_chunk_size))
4  #define blockdist_chunk_end_inv(N, chunk_size, actual_chunk_size) blockdist_chunk_end_inv_r(N,
       ↪ chunk_size, actual_chunk_size, R)
```

Предположим теперь, что при раскладке объектов в корзины требуется пропустить некоторую часть их емкости. В качестве примера рассмотрим размещение данных рассматриваемого массива `A`: $\eta_A(\vec{i}, \vec{z}) = \vec{i}^{(k)} + offset, k = 0, \ldots, p_A - 1$, где $offset$ — пропускаемая часть виртуальных процессоров. Тогда количество объектов в указанной корзине определяется `blockdist_actual_chunk_size_offset_r` (рисунок А.1), что позволяет

определить распределение данных `blockdist_array_distribute_offset_r` (листинг А.5).

Листинг А.5 Размеры порций объектов в корзинах при раскладке со смещением

```
#define triple_choice(test_val, ref_val, eq_val, lt_val, gt_val) (((test_val) == (ref_val)) ?
    ↪ (eq_val) : (((test_val) > (ref_val)) ? (gt_val) : (lt_val)))
#define blockdist_actual_chunk_size_offset_r_int(free_cnt, fst_capacity, chunk_size, N, R)
    ↪ triple_choice(R, free_cnt, fst_capacity, 0, max(0, min((chunk_size), (N) - (fst_capacity) -
    ↪ ((R) - (free_cnt) - 1) * (chunk_size))))
#define blockdist_actual_chunk_size_offset_r(chunk_size, offset, N, R)
    ↪ blockdist_actual_chunk_size_offset_r_int((offset) / (chunk_size), min(N, (chunk_size) -
    ↪ (offset) % (chunk_size)), chunk_size, N, R)
#define blockdist_actual_chunk_size_offset(chunk_size, offset, N)
    ↪ blockdist_actual_chunk_size_offset_r(chunk_size, offset, N, R)
#define blockdist_array_distribute_offset_r(offset, N, R)
    ↪ blockdist_actual_chunk_size_offset_r(block_size, offset, N, R)
#define blockdist_array_distribute_offset(offset, N) blockdist_array_distribute_offset_r(offset, N,
    ↪ R)
```

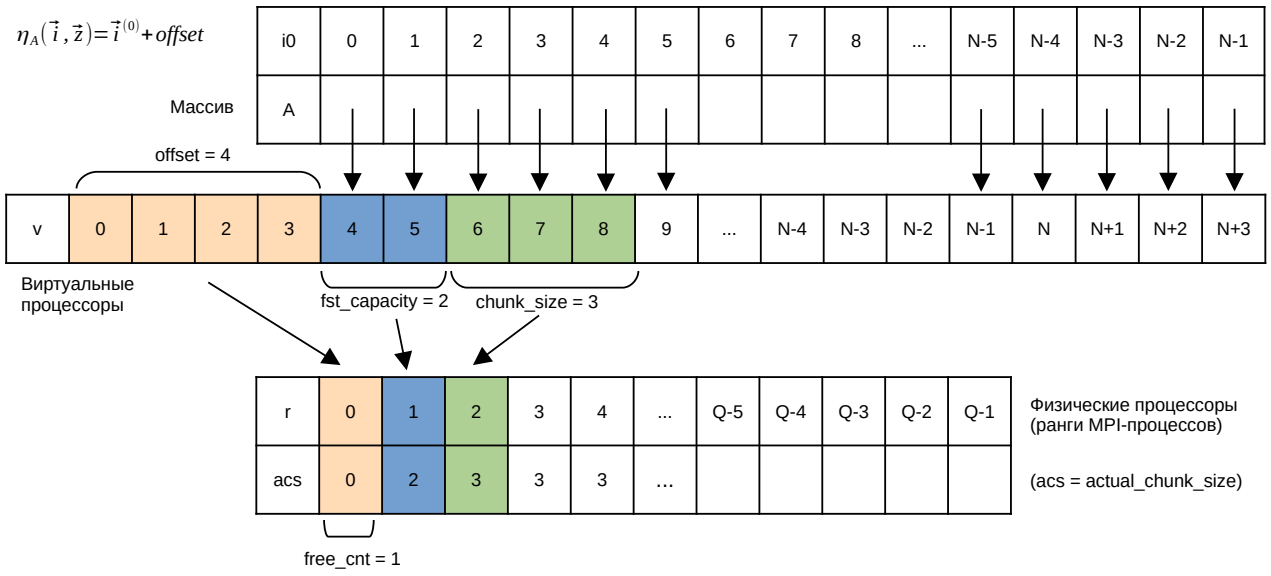


Рисунок А.1 — Вычисление blockdist_actual_chunk_size_offset_r

Можно заметить, что несколько первых корзин могут быть оставлены пустыми (в количестве `free_cnt`), первая занятая может быть заполнена менее остальных (на величину `fst_capacity`), а между всеми занятыми, начиная со второй, объекты будут распределяться как в рассмотренных ранее случаях (в объеме `chunk_size`, кроме последней занятой). Начальные и конечные индексы объектов, попавших в корзины, можно определить применением `blockdist_chunk_start_offset_r` и `blockdist_chunk_end_offset_r` (листинг А.6).

Листинг А.6 Начальные и конечные индексы объектов в корзинах при раскладке со смещением

```
#define blockdist_chunk_start_offset_r(chunk_size, offset, R) max(0, (R) * (chunk_size) - (offset))
#define blockdist_chunk_start_offset(chunk_size, offset) blockdist_chunk_start_offset_r(chunk_size,
    ↪ offset, R)
#define blockdist_chunk_end_offset_r(chunk_size, actual_chunk_size, offset, R)
    ↪ (blockdist_chunk_start_offset_r(chunk_size, offset, R) + (actual_chunk_size))
#define blockdist_chunk_end_offset(chunk_size, actual_chunk_size, offset)
    ↪ blockdist_chunk_end_offset_r(chunk_size, actual_chunk_size, offset, R)
```

А.1.2 Обработка линейных массивов и строк матриц

Рассмотрим передачу информационных пакетов в виде фрагментов линейных массивов. В этом случае предполагается обработка непрерывных участков памяти. Если имеется матрица, линеаризованная по строкам, то ее строки могут рассматриваться как линейные массивы.

Макрос `line_Q_read_send_econd_r_int` выполняет отправку указанного фрагмента линейного массива `arr` процессору `r` для реализации удаленного чтения (листинг А.7). Указываются начальный (`fst_idx`) и конечный (`last_idx`) индексы для фрагмента массива, а также индексы виртуальных процессоров `fst_vp` и `last_vp`, обращающихся к элементам массива на начальной и конечной границах фрагмента соответственно. Для определения принадлежности элементов массива виртуальным процессорам передается параметр `eta`, представляющий найденное размещение данных. Отрезок пространства виртуальных процессоров, обрабатываемый физическим процессором-адресатом `r` вычисляется в границах `lr` и `ur`, имеющих смысл $l(r)$ и $u(r)$ из определений (3.1). В булевой переменной `nonempty` вычисляется результат конъюнкции следующих условий: индекс конечной границы фрагмента не меньше индекса начальной границы, элементами на границах фрагмента владеет физический процессор-отправитель `R`, индексы виртуальных процессоров, обращающихся к граничным элементам фрагмента, сопоставлены физическому процессору-адресату. В силу непрерывности отрезка виртуальных процессоров, сопоставленных физическому, для того, чтобы удостовериться в принадлежности фрагмента процессору-отправителю, достаточно проверить размещение только граничных элементов фрагмента. По этой же причине нет необходимости проверять принадлежность всех индексов виртуальных процессоров между `fst_vp` и `last_vp` процессору-адресату.

Также введен пользовательский конъюнкт `econd` для отладочных целей в качестве параметра, значение которого должно быть истинным в нормальной работе. Только в случае истинности `nonempty` считается, что многогранник коммуникаций непустой, и выполняется неблокирующая отправка сообщения MPI с тегом `tag`. Перебор всех потенциальных процессоров-адресатов в коммуникаторе `MPI_COMM_WORLD` выполняется макросами `line_Q_read_send_econd_int` (для общего случая) и `line_Q_read_send_econd_fco_int` (для случая, когда выполняется свойство вперед направленных коммуникаций). Процессор-отправитель всегда пропускается, так как нет смысла в информационном обмене, когда отправитель и адресат совпадают.

Листинг А.7 Удаленное чтение: отправка фрагмента линейного массива

```
#define line_Q_read_send_econd_r_int(fst_idx, fst_vp, last_idx, last_vp, econd, arr, eta, tag, r) { \
    int r_actual_chunk_size = blockdist_actual_chunk_size_r(block_size, blockdist_nb_vproc, r); \
    int lr = L + r * block_size; \
    int ur = lr + r_actual_chunk_size - 1; \
    bool nonempty = (econd) && ((last_idx) >= (fst_idx)) && \
          (blockdist_proc_rank(eta(fst_idx)) == R) && (blockdist_proc_rank(eta(last_idx)) == R) && \
          ((fst_vp) >= lr && (fst_vp) <= ur) && ((last_vp) >= lr && (last_vp) <= ur); \
    if (nonempty) { \
      MPI_Request request; \
      MPI_Isend((arr) + (fst_idx), (last_idx) - (fst_idx) + 1, MPI_DOUBLE, r, tag, MPI_COMM_WORLD,
    ↪ &request); \
      MPI_Request_free(&request); \
    } \
  }
#define line_Q_read_send_econd_int(fst_idx, fst_vp, last_idx, last_vp, econd, arr, eta, tag) \
  for (int r = 0; r < Q; r++) { \
    if (r != R) { \
      line_Q_read_send_econd_r_int(fst_idx, fst_vp, last_idx, last_vp, econd, arr, eta, tag, r); \
    } \
  }
#define line_Q_read_send_econd_fco_int(fst_idx, fst_vp, last_idx, last_vp, econd, arr, eta, tag) \
  for (int r = R + 1; r < Q; r++) { \
    line_Q_read_send_econd_r_int(fst_idx, fst_vp, last_idx, last_vp, econd, arr, eta, tag, r); \
  }
```

Для выполнения экспериментов предпочтительны `line_Q_read_send` и `line_Q_read_send_fco`, поддерживающие измерение времени выполнения передачи, и фиксирующие `econd == true` (листинг А.8).

Макрос `line_Q_read_receive_econd_r_int` выполняет прием указанного фрагмента линейного массива `arr` от процессора `r` для реализации удаленного чтения (листинг А.9). Вычисление `nonempty` отличается следующими условиями: элементами на границах фрагмента владеет физический процессор-отправитель

Листинг А.8 Удаленное чтение: обертки для отправки данных

```
#define line_Q_read_send(fst_idx, fst_vp, last_idx, last_vp, arr, eta, tag) \
  IF_TIME(mpi_spent_time -= rtclock()); \
  line_Q_read_send_econd_int(fst_idx, fst_vp, last_idx, last_vp, true, arr, eta, tag); \
  IF_TIME(mpi_spent_time += rtclock());
#define line_Q_read_send_fco(fst_idx, fst_vp, last_idx, last_vp, arr, eta, tag) \
  IF_TIME(mpi_spent_time -= rtclock()); \
  line_Q_read_send_econd_fco_int(fst_idx, fst_vp, last_idx, last_vp, true, arr, eta, tag); \
  IF_TIME(mpi_spent_time += rtclock())
```

`r`, индексы виртуальных процессоров, обращающихся к граничным элементам фрагмента, сопоставлены физическому процессору-адресату `R`.

Листинг А.9 Удаленное чтение: прием фрагмента линейного массива

```
#define line_Q_read_receive_econd_r_int(fst_idx, fst_vp, last_idx, last_vp, econd, arr, eta, tag, r)
    ↪ { \
    bool nonempty = (econd) && ((last_idx) >= (fst_idx)) && \
          (blockdist_proc_rank(eta(fst_idx)) == r) && (blockdist_proc_rank(eta(last_idx)) == r) && \
          blockdist_in_range(fst_vp) && blockdist_in_range(last_vp); \
    if (nonempty) { \
      MPI_Status status; \
      MPI_Recv((arr) + (fst_idx), (last_idx) - (fst_idx) + 1, MPI_DOUBLE, r, tag, MPI_COMM_WORLD,
    ↪ &status); \
    } \
  }
#define line_Q_read_receive_econd_int(fst_idx, fst_vp, last_idx, last_vp, econd, arr, eta, tag) \
  for (int r = 0; r < Q; r++) { \
    if (r != R) { \
      line_Q_read_receive_econd_r_int(fst_idx, fst_vp, last_idx, last_vp, econd, arr, eta, tag, r); \
    } \
  }
#define line_Q_read_receive_econd_fco_int(fst_idx, fst_vp, last_idx, last_vp, econd, arr, eta, tag) \
  for (int r = 0; r < R; r++) { \
    line_Q_read_receive_econd_r_int(fst_idx, fst_vp, last_idx, last_vp, econd, arr, eta, tag, r); \
  }
```

Макросы `line_Q_read_receive` и `line_Q_read_receive_fco` реализуются так же, как их аналоги `line_Q_read_send` и `line_Q_read_send_fco`.

Макрос `line_Q_write_send_econd_r_int` выполняет отправку указанного фрагмента линейного массива `arr` процессору `r` для реализации удаленной записи (листинг А.10). Вычисление `nonempty` полностью совпадает с `line_Q_read_receive_econd_r_int`.

Макрос `line_Q_write_receive_econd_r_int` выполняет прием указанного фрагмента линейного массива `arr` от процессора `r` для реализации удаленной записи (листинг А.11). Вычисление `nonempty` полностью совпадает с `line_Q_read_send_econd_r_int`.

Листинг А.10 Удаленная запись: отправка фрагмента линейного массива

```
#define line_Q_write_send_econd_r_int(fst_idx, fst_vp, last_idx, last_vp, econd, arr, eta, tag, r) { \
    bool nonempty = (econd) && ((last_idx) >= (fst_idx)) && \
          (blockdist_proc_rank(eta(fst_idx)) == r) && (blockdist_proc_rank(eta(last_idx)) == r) && \
          blockdist_in_range(fst_vp) && blockdist_in_range(last_vp); \
    if (nonempty) { \
      MPI_Request request; \
      MPI_Isend((arr) + (fst_idx), (last_idx) - (fst_idx) + 1, MPI_DOUBLE, r, tag, MPI_COMM_WORLD, &request); \
      MPI_Request_free(&request); \
    } \
  }
#define line_Q_write_send_econd_int(fst_idx, fst_vp, last_idx, last_vp, econd, arr, eta, tag) \
  for (int r = 0; r < Q; r++) { \
    if (r != R) { \
      line_Q_write_send_econd_r_int(fst_idx, fst_vp, last_idx, last_vp, econd, arr, eta, tag, r); \
    } \
  }
#define line_Q_write_send_econd_fco_int(fst_idx, fst_vp, last_idx, last_vp, econd, arr, eta, tag) \
  for (int r = R + 1; r < Q; r++) { \
    line_Q_write_send_econd_r_int(fst_idx, fst_vp, last_idx, last_vp, econd, arr, eta, tag, r); \
  }
```

Листинг А.11 Удаленная запись: прием фрагмента линейного массива

```
#define line_Q_write_receive_econd_r_int(fst_idx, fst_vp, last_idx, last_vp, econd, arr, eta, tag, r) { \
    int r_actual_chunk_size = blockdist_actual_chunk_size_r(block_size, blockdist_nb_vproc, r); \
    int lr = L + r * block_size; \
    int ur = lr + r_actual_chunk_size - 1; \
    bool nonempty = (econd) && ((last_idx) >= (fst_idx)) && \
          (blockdist_proc_rank(eta(fst_idx)) == R) && (blockdist_proc_rank(eta(last_idx)) == R) && \
          ((fst_vp) >= lr && (fst_vp) <= ur) && ((last_vp) >= lr && (last_vp) <= ur); \
    if (nonempty) { \
      MPI_Status status; \
      MPI_Recv((arr) + (fst_idx), (last_idx) - (fst_idx) + 1, MPI_DOUBLE, r, tag, MPI_COMM_WORLD, &status); \
    } \
  }
#define line_Q_write_receive_econd_int(fst_idx, fst_vp, last_idx, last_vp, econd, arr, eta, tag) \
  for (int r = 0; r < Q; r++) { \
    if (r != R) { \
      line_Q_write_receive_econd_r_int(fst_idx, fst_vp, last_idx, last_vp, econd, arr, eta, tag, r); \
    } \
  }
#define line_Q_write_receive_econd_fco_int(fst_idx, fst_vp, last_idx, last_vp, econd, arr, eta, tag) \
  for (int r = 0; r < R; r++) { \
    line_Q_write_receive_econd_r_int(fst_idx, fst_vp, last_idx, last_vp, econd, arr, eta, tag, r); \
  }
```

Макросы `line_Q_write_send` и `line_Q_write_receive`, а также их варианты для случая вперед направленных коммуникаций `line_Q_write_send_fco`

и `line_Q_write_receive_fco`, реализуются так же, как их аналоги, реализующие удаленное чтение.

На основе разработанных макросов созданы конструкции, реализующие операции удаленного доступа для нескольких частных случаев. Рассмотрим инструкцию с удаленным доступом внутри параллельного цикла, пусть его индекс имеет имя `ilpp` и итерирует по пространству виртуальных процессоров. Предположим, что цикл по `ilpp` — наиболее глубоко вложенный цикл в гнезде, включающем инструкцию с удаленным доступом. В силу того, что индексная функция доступа к массиву является аффинной, арифметическое выражение (индекс массива) можно представить в виде `mul * ilpp + add`, где `mul` — целочисленная неотрицательная константа, `add` — аффинная форма, включающая индексы других циклов, отсчитывающих логическое время, и внешние переменные программы. Макросы `line_Q_read_vp` и `line_Q_write_vp` реализуют удаленное чтение и удаленную запись для рассмотренного случая (листинг А.12).

Листинг А.12 Удаленное чтение и удаленная запись для `mul` $\geqslant 0$

```
#define clamp(v, l, u) min(max(v, l), u)
#define line_Q_read_vp_econd(mul, add, lb, ub, econd, arr, eta, tag) \
  IF_TIME(mpi_spent_time -= rtclock()); \
  line_Q_read_send_econd_int((mul) * clamp(lr, lb, ub) + (add), clamp(lr, lb, ub), (mul) * clamp(ur,
    ↪ lb, ub) + (add), clamp(ur, lb, ub), \
                econd, arr, eta, tag); \
  line_Q_read_receive_econd_int((mul) * clamp(lR, lb, ub) + (add), clamp(lR, lb, ub), (mul) *
    ↪ clamp(uR, lb, ub) + (add), clamp(uR, lb, ub), \
                 econd, arr, eta, tag); \
  MPI_Barrier(MPI_COMM_WORLD); \
  IF_TIME(mpi_spent_time += rtclock());
#define line_Q_read_vp(mul, add, lb, ub, arr, eta, tag) line_Q_read_vp_econd(mul, add, lb, ub, true,
    ↪ arr, eta, tag)
#define line_Q_write_vp_econd(mul, add, lb, ub, econd, arr, eta, tag) \
  IF_TIME(mpi_spent_time -= rtclock()); \
  line_Q_write_send_econd_int((mul) * clamp(lR, lb, ub) + (add), clamp(lR, lb, ub), (mul) *
    ↪ clamp(uR, lb, ub) + (add), clamp(uR, lb, ub), \
                econd, arr, eta, tag); \
  line_Q_write_receive_econd_int((mul) * clamp(lr, lb, ub) + (add), clamp(lr, lb, ub), (mul) *
    ↪ clamp(ur, lb, ub) + (add), clamp(ur, lb, ub), \
                 econd, arr, eta, tag); \
  MPI_Barrier(MPI_COMM_WORLD); \
  IF_TIME(mpi_spent_time += rtclock());
#define line_Q_write_vp(mul, add, lb, ub, arr, eta, tag) line_Q_write_vp_econd(mul, add, lb, ub,
    ↪ true, arr, eta, tag)
```

Параметры `lb` и `ub` представляют нижнюю и верхнюю границу изменения счетчика `ilpp`, а макрос `clamp` позволяет ограничить для каждого вовлекаемого в информационный обмен физического процессора сопоставленный ему отрезок

пространства виртуальных процессоров только теми из них, что относятся к параллельному циклу.

В случае, если `mul` — отрицательная целочисленная константа, индексы виртуальных процессоров, соответствующих начальной и конечной границам фрагмента массива, меняются местами, что отражено в реализациях макросов `line_Q_read_vp_rev` и `line_Q_write_vp_rev` (листинг А.13).

Листинг А.13 Удаленное чтение и удаленная запись для `mul` < 0

```
#define line_Q_read_vp_rev_econd(mul, add, lb, ub, econd, arr, eta, tag) \
  IF_TIME(mpi_spent_time -= rtclock()); \
  line_Q_read_send_econd_int((mul) * clamp(ur, lb, ub) + (add), clamp(ur, lb, ub), (mul) * clamp(lr,
    ↪ lb, ub) + (add), clamp(lr, lb, ub), \
               econd, arr, eta, tag); \
  line_Q_read_receive_econd_int((mul) * clamp(uR, lb, ub) + (add), clamp(uR, lb, ub), (mul) *
    ↪ clamp(lR, lb, ub) + (add), clamp(lR, lb, ub), \
                econd, arr, eta, tag); \
  MPI_Barrier(MPI_COMM_WORLD); \
  IF_TIME(mpi_spent_time += rtclock());
#define line_Q_read_vp_rev(mul, add, lb, ub, arr, eta, tag) line_Q_read_vp_rev_econd(mul, add, lb,
    ↪ ub, true, arr, eta, tag)
#define line_Q_write_vp_rev_econd(mul, add, lb, ub, econd, arr, eta, tag) \
  IF_TIME(mpi_spent_time -= rtclock()); \
  line_Q_write_send_econd_int((mul) * clamp(uR, lb, ub) + (add), clamp(uR, lb, ub), (mul) *
    ↪ clamp(lR, lb, ub) + (add), clamp(lR, lb, ub), \
               econd, arr, eta, tag); \
  line_Q_write_receive_econd_int((mul) * clamp(ur, lb, ub) + (add), clamp(ur, lb, ub), (mul) *
    ↪ clamp(lr, lb, ub) + (add), clamp(lr, lb, ub), \
                econd, arr, eta, tag); \
  MPI_Barrier(MPI_COMM_WORLD); \
  IF_TIME(mpi_spent_time += rtclock());
#define line_Q_write_vp_rev(mul, add, lb, ub, arr, eta, tag) line_Q_write_vp_rev_econd(mul, add, lb,
    ↪ ub, true, arr, eta, tag)
```

А.1.3 Обработка столбцов матриц

Рассмотрим передачу информационных пакетов, обертывающих столбцы матриц, линеаризованных по строкам. В этом случае предполагается перебор подмножества строк матрицы для составления информационного пакета.

Макрос `col_Q_read_send_econd_r_int` выполняет отправку указанного фрагмента столбца матрицы `arr` процессору `r` для реализации удаленного чтения (листинг А.14). Индекс столбца передается параметром `col_idx`, также указываются начальный (`fst_row`) и конечный (`last_row`) индексы строк

для фрагмента столбца, а также индексы виртуальных процессоров `fst_vp` и `last_vp`, обращающихся к элементам столбца на начальной и конечной границах фрагмента соответственно. Для определения принадлежности элементов массива виртуальным процессорам передается параметр `eta`, представляющий найденное размещение данных. Дополнительно передается указатель на буфер `buf`, в котором формируется информационный пакет (требование к вместимости: буфер должен вмещать столбец матрицы целиком). Вычисление `nonempty` сходно с таковым у `line_Q_read_send_econd_r_int`, разница только в макросе `eta`, принимающем на вход оба индекса двумерного массива. Если многогранник коммуникаций непустой, то выполняется копирование фрагмента столбца в буфер, и затем происходит неблокирующая отправка.

Листинг А.14 Удаленное чтение: отправка фрагмента столбца матрицы

```
#define col_Q_read_send_econd_r_int(fst_row, fst_vp, last_row, last_vp, col_idx, econd, buf, arr, eta, tag, r) { \
    int r_actual_chunk_size = blockdist_actual_chunk_size_r(block_size, blockdist_nb_vproc, r); \
    int lr = L + r * block_size; \
    int ur = lr + r_actual_chunk_size - 1; \
    bool nonempty = (econd) && ((last_row) >= (fst_row)) && \
          (blockdist_proc_rank(eta(fst_row, col_idx)) == R) && (blockdist_proc_rank(eta(last_row, col_idx)) == R) && \
          ((fst_vp) >= lr && (fst_vp) <= ur) && ((last_vp) >= lr && (last_vp) <= ur); \
    if (nonempty) { \
      for (int i = fst_row; i <= last_row; i++) { \
        buf[i] = arr[i][col_idx]; \
      } \
      MPI_Request request; \
      MPI_Isend((buf) + (fst_row), (last_row) - (fst_row) + 1, MPI_DOUBLE, r, tag, MPI_COMM_WORLD, &request); \
      MPI_Request_free(&request); \
    } \
  }
```

Макрос `col_Q_read_receive_econd_r_int` выполняет прием указанного фрагмента столбца матрицы `arr` от процессора `r` для реализации удаленного чтения (листинг А.15). Вычисление `nonempty` сходно с таковым у `line_Q_read_receive_econd_r_int`, разница только в макросе `eta`, принимающем на вход оба индекса двумерного массива. Если многогранник коммуникаций непустой, то после приема информационного пакета в буфер выполняется копирование фрагмента столбца из буфера в матрицу.

Макрос `col_Q_write_send_econd_r_int` выполняет отправку указанного фрагмента столбца матрицы `arr` процессору `r` для реализации удален-

Листинг А.15 Удаленное чтение: прием фрагмента столбца матрицы

```
#define col_Q_read_receive_econd_r_int(fst_row, fst_vp, last_row, last_vp, col_idx, econd, buf, arr,
    ↪ eta, tag, r) { \
    bool nonempty = (econd) && ((last_row) >= (fst_row)) && \
          (blockdist_proc_rank(eta(fst_row, col_idx)) == r) && (blockdist_proc_rank(eta(last_row,
    ↪ col_idx)) == r) && \
          blockdist_in_range(fst_vp) && blockdist_in_range(last_vp); \
    if (nonempty) { \
      MPI_Status status; \
      MPI_Recv((buf) + (fst_row), (last_row) - (fst_row) + 1, MPI_DOUBLE, r, tag, MPI_COMM_WORLD,
    ↪ &status); \
      for (int i = fst_row; i <= last_row; i++) { \
        arr[i][col_idx] = buf[i]; \
      } \
    } \
  }
```

ной записи (листинг А.16). Вычисление `nonempty` полностью совпадает с `col_Q_read_receive_econd_r_int`.

Листинг А.16 Удаленная запись: отправка фрагмента столбца матрицы

```
#define col_Q_write_send_econd_r_int(fst_row, fst_vp, last_row, last_vp, col_idx, econd, buf, arr,
    ↪ eta, tag, r) { \
    bool nonempty = (econd) && ((last_row) >= (fst_row)) && \
          (blockdist_proc_rank(eta(fst_row, col_idx)) == r) && (blockdist_proc_rank(eta(last_row,
    ↪ col_idx)) == r) && \
          blockdist_in_range(fst_vp) && blockdist_in_range(last_vp); \
    if (nonempty) { \
      for (int i = fst_row; i <= last_row; i++) { \
        buf[i] = arr[i][col_idx]; \
      } \
      MPI_Request request; \
      MPI_Isend((buf) + (fst_row), (last_row) - (fst_row) + 1, MPI_DOUBLE, r, tag, MPI_COMM_WORLD,
    ↪ &request); \
      MPI_Request_free(&request); \
    } \
  }
```

Макрос `col_Q_write_receive_econd_r_int` выполняет прием указанного фрагмента столбца матрицы `arr` от процессора `r` для реализации удаленной записи (листинг А.17). Вычисление `nonempty` полностью совпадает с `col_Q_read_send_econd_r_int`.

Макросы для удаленного чтения (`col_Q_read_send`, `col_Q_read_receive`, `col_Q_read_vp`, `col_Q_read_vp_rev`) наряду с макросами для удаленной записи (`col_Q_write_send`, `col_Q_write_receive`, `col_Q_write_vp`, `col_Q_write_vp_rev`), а также все их варианты с суффиксом `_fco` реализуются так же, как и их аналоги с префиксом `line_`.

Листинг А.17 Удаленная запись: прием фрагмента столбца матрицы

```
#define col_Q_write_receive_econd_r_int(fst_row, fst_vp, last_row, last_vp, col_idx, econd, buf,
    ↪ arr, eta, tag, r) { \
    int r_actual_chunk_size = blockdist_actual_chunk_size_r(block_size, blockdist_nb_vproc, r); \
    int lr = L + r * block_size; \
    int ur = lr + r_actual_chunk_size - 1; \
    bool nonempty = (econd) && ((last_row) >= (fst_row)) && \
          (blockdist_proc_rank(eta(fst_row, col_idx)) == R) && (blockdist_proc_rank(eta(last_row,
    ↪ col_idx)) == R) && \
          ((fst_vp) >= lr && (fst_vp) <= ur) && ((last_vp) >= lr && (last_vp) <= ur); \
    if (nonempty) { \
      MPI_Status status; \
      MPI_Recv((buf) + (fst_row), (last_row) - (fst_row) + 1, MPI_DOUBLE, r, tag, MPI_COMM_WORLD,
    ↪ &status); \
      for (int i = fst_row; i <= last_row; i++) { \
        arr[i][col_idx] = buf[i]; \
      } \
    } \
  }
```

А.1.4 *Локализация результатов вычислений*

По завершению распределенных вычислений результат в виде одномерных и двумерных массивов может остаться распределенным по множеству параллельно работающих процессов в согласии с вычисленным размещением данных. Это может оказаться неудобным для его дальнейшего использования. Для нескольких частных случаев, далее реализованных в экспериментах, разработаны функции для локализации результатов вычислений в виде матриц в памяти процесса с рангом 0. Линейный массив может рассматриваться как матрица с одной строкой.

Пусть размещение рассматриваемого массива `A` имеет вид $\eta_A(\vec{i}, \vec{z}) = \vec{i}^{(0)}$. Тогда функция `collect_matrix_rows_double` (листинг А.18) может быть использована для того, чтобы собрать строки матрицы `m` со всех процессов в коммуникаторе `MPI_COMM_WORLD` в процессе с рангом 0.

Если размещение рассматриваемого массива `A` имеет вид $\eta_A(\vec{i}, \vec{z}) = \vec{i}^{(1)}$ или $\eta_A(\vec{i}, \vec{z}) = \vec{z}^{(1)} - \vec{i}^{(1)}$, то функция `collect_matrix_cols_double` (листинг А.19) может быть использована для того, чтобы собрать столбцы матрицы `m` со всех процессов в коммуникаторе `MPI_COMM_WORLD` в процессе с рангом 0. Параметр `inv` указывает на обратный порядок укладки столбцов матрицы по виртуальным процессорам, если имеет значение `true`.

Листинг А.18 Сбор строк матрицы со всех процессов в коммуникаторе

```
void collect_matrix_rows_double(double** m, int rows, int cols, int chunk_size) {
  IF_TIME(mpi_spent_time -= rtclock());
  if (R == 0) {
    for (int r = 1; r < Q; r++) {
      int actual_chunk_size_r = blockdist_actual_chunk_size_r(chunk_size, rows, r);
      for (int i = blockdist_chunk_start_r(chunk_size, r); i < blockdist_chunk_end_r(chunk_size,
    ↪ actual_chunk_size_r, r); i++) {
        MPI_Status status;
        MPI_Recv(m[i], cols, MPI_DOUBLE, r, i, MPI_COMM_WORLD, &status);
      }
    }
  }
  else {
    int actual_chunk_size = blockdist_actual_chunk_size(chunk_size, rows);
    for (int i = blockdist_chunk_start(chunk_size); i < blockdist_chunk_end(chunk_size,
    ↪ actual_chunk_size); i++) {
      MPI_Send(m[i], cols, MPI_DOUBLE, 0, i, MPI_COMM_WORLD);
    }
  }
  MPI_Barrier(MPI_COMM_WORLD);
  IF_TIME(mpi_spent_time += rtclock());
}
```

Листинг А.19 Сбор столбцов матрицы со всех процессов в коммуникаторе

```
void collect_matrix_cols_double(double** m, int rows, int cols, int chunk_size, bool inv) {
  IF_TIME(mpi_spent_time -= rtclock());
  if (R == 0) {
    for (int i = 0; i < rows; i++) {
      for (int r = 1; r < Q; r++) {
        int receive_chunk_size = blockdist_actual_chunk_size_r(chunk_size, cols, r);
        if (receive_chunk_size > 0) {
          MPI_Status status;
          int offset = (inv) ? blockdist_chunk_start_inv_r(cols, chunk_size, receive_chunk_size, r) :
                    blockdist_chunk_start_r(chunk_size, r);
          MPI_Recv(m[i] + offset, receive_chunk_size, MPI_DOUBLE, r, i, MPI_COMM_WORLD, &status);
        }
      }
    }
  }
  else {
    int send_chunk_size = blockdist_actual_chunk_size(chunk_size, cols);
    if (send_chunk_size > 0) {
      for (int i = 0; i < rows; i++) {
        int offset = (inv) ? blockdist_chunk_start_inv(cols, chunk_size, send_chunk_size) :
                  blockdist_chunk_start(chunk_size);
        MPI_Send(m[i] + offset, send_chunk_size, MPI_DOUBLE, 0, i, MPI_COMM_WORLD);
      }
    }
  }
  MPI_Barrier(MPI_COMM_WORLD);
  IF_TIME(mpi_spent_time += rtclock());
}
```

Если размещение рассматриваемого массива A имеет вид $\eta_A(\vec{i}, \vec{z}) = \vec{i}^{(1)} + offset$, то функция `collect_matrix_cols_double_offset` (листинг А.20) может быть использована для того, чтобы собрать столбцы матрицы `m` со всех процессов в коммуникаторе `MPI_COMM_WORLD` в процессе с рангом 0. Параметр `offset` указывает на количество виртуальных процессоров, пропущенных при укладке столбцов матрицы.

Листинг А.20 Сбор столбцов матрицы со всех процессов в коммуникаторе при раскладке со смещением

```
1  void collect_matrix_cols_double_offset(double** m, int rows, int cols, int chunk_size, int offset) {
2    IF_TIME(mpi_spent_time -= rtclock());
3    if (R == 0) {
4      for (int i = 0; i < rows; i++) {
5        for (int r = 1; r < Q; r++) {
6          int receive_chunk_size = blockdist_actual_chunk_size_offset_r(chunk_size, offset, cols, r);
7          if (receive_chunk_size > 0) {
8            MPI_Status status;
9            MPI_Recv(m[i] + blockdist_chunk_start_offset_r(chunk_size, offset, r), receive_chunk_size,
     ↪ MPI_DOUBLE, r, i, MPI_COMM_WORLD, &status);
10         }
11       }
12     }
13   }
14   else {
15     int send_chunk_size = blockdist_actual_chunk_size_offset(chunk_size, offset, cols);
16     if (send_chunk_size > 0) {
17       for (int i = 0; i < rows; i++) {
18         MPI_Send(m[i] + blockdist_chunk_start_offset(chunk_size, offset), send_chunk_size,
     ↪ MPI_DOUBLE, 0, i, MPI_COMM_WORLD);
19       }
20     }
21   }
22   MPI_Barrier(MPI_COMM_WORLD);
23   IF_TIME(mpi_spent_time += rtclock());
24 }
```

А.2 Скрипты сборки и запуска приложений

Листинг А.21 Общий файл сборки для OpenMP omp.mk

```
PLUTO_HOME=$(HOME)/pluto-distmem/pluto

suffix=pluto
pluto_c?=$(alg).$(suffix).c
5 pluto_cloog?=$(alg).$(suffix).cloog
```

```
pluto: $(pluto_c)
%.$(suffix).c: ../../%.c
  $(PLUTO_HOME)/polycc $< --parallelize --lastwriter -o $@ && \
  mv $*.$(suffix).pluto.cloog $*.$(suffix).cloog
clean_pluto:
  rm -f $(pluto_c) $(pluto_cloog)

alg_c?=$(alg).c
cflags?=-O3 -std=c99 -fopenmp

$(alg): $(alg_c)
  gcc $(cflags) $^ ../common/src/lcg.c ../common/src/util.c -o $@ -I../common/include -lm
clean:
  rm -f $(alg)
```

Листинг А.22 Файл Makefile сборки lu для OpenMP

```
alg=lu
all: $(alg)
include ../common/make/omp.mk
```

Листинг А.23 Файл Makefile сборки atax для OpenMP

```
alg=atax
pluto_c=atax.pluto.c atax_p.pluto.c
pluto_cloog=atax.pluto.cloog atax_p.pluto.cloog
all: $(alg)
include ../common/make/omp.mk
```

Листинг А.24 Файл Makefile сборки syr2k для OpenMP

```
alg=syr2k
all: $(alg)
include ../common/make/omp.mk
```

Листинг А.25 Файл Makefile сборки floyd для OpenMP

```
alg=floyd
all: $(alg)
include ../common/make/omp.mk
```

Листинг А.26 Файл Makefile сборки gramschmidt для OpenMP

```
alg=gramschmidt
all: $(alg)
include ../common/make/omp.mk
```

Листинг А.27 Общий файл сборки для MPI mpi.mk

```
PLUTO_HOME=$(HOME)/pluto-distmem/pluto

suffix=pluto_distmem
pluto_c=$(alg).$(suffix).c pi_$(alg).$(suffix).c sigma_$(alg).$(suffix).c
pluto_h=$(alg).$(suffix).h pi_defs.h
pluto_cloog=$(alg).$(suffix).cloog sigma_fop.cloog write_out.cloog is_receiver_fop.cloog

pluto: $(pluto_c)
pi_%.$(suffix).c: %.$(suffix).c
```

```
sigma_%.$(suffix).c: %.$(suffix).c
%.$(suffix).c: %.pluto_wrapper.c
  $(PLUTO_HOME)/polycc $< --distmem --commopt_fop --isldep --lastwriter --cloogsh
    ↪ --timereport -o $@ && \
  mv $*.$(suffix).pluto.cloog $*.$(suffix).cloog && \
  sed -i $*.$(suffix).c -e 's/MPI_Init/\/\/MPI_init/g;s/MPI_Finalize/\/\/MPI_Finalize/g' \
    -e 's/fopen(".*", "r")/0/' \
    -e 's/IF_TIME(t_local = rtclock());/MPI_Barrier(MPI_COMM_WORLD); IF_TIME(t_local =
    ↪ rtclock());/' \
    -e 's/IF_TIME(t_local = rtclock() - t_local);/MPI_Barrier(MPI_COMM_WORLD);
    ↪ IF_TIME(t_local = rtclock() - t_local); IF_TIME(pluto_spent_time = t_local);
    ↪ IF_TIME(mpi_spent_time = t_comm + t_pack + t_unpack);/'
%.pluto_wrapper.c: ../../%.c
  echo -e "#include <stdio.h>\n#include <util.h>\n#include <umacros.h>\n" > $@ && \
  echo -e "extern double mpi_spent_time;\nextern double pluto_spent_time;\n" >> $@ && \
  echo -e "$(global_variables)" >> $@ && \
  echo -e "void $(alg)_pluto_mpi() {" >> $@ && \
  sed $< -e "$(sed_preprocess_expr)" >> $@ && \
  echo -e "\n}\n" >> $@
clean_pluto:
  rm -f $(pluto_c) $(pluto_h) $(pluto_cloog)

$(alg)_mpi: $(alg)_mpi.c $(pluto_c)
  mpicc -O3 -std=c99 -fopenmp $(pluto_defines) $^ ../common/src/lcg.c ../common/src/util.c
    ↪ $(PLUTO_HOME)/polyrt/polyrt.c -o $@ -I../common/include -I$(PLUTO_HOME)/polyrt -lm
clean:
  rm -f $(alg)_mpi
```

Листинг А.28 Файл Makefile сборки lu_mpi для MPI

```
alg=lu
pluto_defines=-D_GNU_SOURCE
global_variables=int N;\ndouble** A;\n
all: $(alg)_mpi
time: pluto_defines+=-DTIME
time: all
include ../common/make/mpi.mk
```

Листинг А.29 Файл Makefile сборки atax_p_mpi для MPI

```
alg=atax_p
pluto_defines=-D_GNU_SOURCE
global_variables=int M;\nint N;\ndouble** A;\ndouble* x;\ndouble* y;\ndouble* tmp;\n
all: $(alg)_mpi
time: pluto_defines+=-DTIME
time: all
include ../common/make/mpi.mk
```

Листинг А.30 Файл Makefile сборки syr2k_mpi для MPI

```
alg=syr2k
pluto_defines=-D_GNU_SOURCE
global_variables=int N;\nint M;\ndouble alpha;\ndouble beta;\ndouble** A;\ndouble** B;\ndouble** C;\n
sed_preprocess_expr=s/C\[i\]\[j\] += C\[i\]\[j\];/C\[i\]\[j\] *= beta;/g;s/C\[i\]\[j\] += A\[j\]\[k
    \]\*B\[i\]\[k\] + B\[j\]\[k\]\*A\[i\]\[k\]/C\[i\]\[j\] += alpha * (A\[j\]\[k\]*B\[i\]\[k\] + B\[j
    \]\[k\]*A\[i\]\[k\])/g
all: $(alg)_mpi
time: pluto_defines+=-DTIME
time: all
include ../common/make/mpi.mk
```

Листинг А.31 Файл Makefile сборки floyd_mpi для MPI

```
alg=floyd
pluto_defines=-D_GNU_SOURCE
global_variables=int N;\ndouble** A;\n
sed_preprocess_expr=s/C\[i\]\[j\] += C\[i\]\[j\];/C\[i\]\[j\] *= beta;/g;s/A\[i\]\[j\] = A\[i\]\[k\]
    + A\[k\]\[j\] + A\[i\]\[j\]/A\[i\]\[j\] = min(A\[i\]\[k\] + A\[k\]\[j\], A\[i\]\[j\])/g
all: $(alg)_mpi
time: pluto_defines+=-DTIME
time: all
include ../common/make/mpi.mk
```

Листинг А.32 Файл Makefile сборки gramschmidt_mpi для MPI

```
alg=gramschmidt
pluto_defines=-D_GNU_SOURCE
global_variables=int M;\nint N;\ndouble** A;\ndouble** Q;\ndouble** R;\n
sed_preprocess_expr=s/R\[k\]\[k\] = nrm/R\[k\]\[k\] = sqrt(nrm)/g
all: $(alg)_mpi
time: pluto_defines+=-DTIME
time: all
include ../common/make/mpi.mk
```

Листинг А.33 Файл отправки в SLURM заданий OpenMP omp_bench.sh

```
#!/bin/bash

export PASS_COUNT=10

for t in lu atax syr2k floyd gramschmidt; do
  echo "Running $t..." && \
  d=`pwd`/$t && \
  sbatch --exclusive -x node1 -c 10 -n 1 -o $t.log -D $d $t.sh
done
```

Листинг А.34 Общий файл отправки в SLURM задания MPI submit.sh

```
submit() {
  procs=$((nodes*tasks))
  if [ "$procs" -ne "0" ]; then
    prev_IFS=$IFS
    local IFS=x
    logfile=${prog}_$*_${method}_${procs}_${nodes}x${tasks}.log
    local IFS=$prev_IFS
    if [ ! -f "$logfile" ]; then
      touch $logfile
      sbatch --exclusive --exclude node1 --cpus-per-task=1 --ntasks-per-node=$tasks --nodes $nodes -o
     $logfile -D `dirname "$0"` $prog.script $procs $@ $method
    else
      echo "$logfile already exists."
    fi
  fi
}
```

A.3 Реализация линейного конгруэнтного генератора

Листинг A.35 lcg.h

```c
#ifndef LCG_H
#define LCG_H

// Uniform random function code is based on linear congruential generator
// http://www.cs.wm.edu/~va/software/park/park.html

#define MODULUS 2147483647
#define MULTIPLIER 48271

/**
 * Returns a pseudo-random real number uniformly distributed between 0 and 1.
 */
double linear_congruential_gen(int* seed);
/**
 * Fills the fragment from i1 to i2 index (non-including) of the vector v with pseudo-random values.
 */
void fill_vector_double(double* v, int len, int i1, int i2, int* seed);
/**
 * Fills the fragment from r1 row to r2 row (non-including) and from c1 column to c2 column
    ↪ (non-including) of the matrix m with pseudo-random values.
 */
void fill_matrix_double(double** m, int rows, int cols, int r1, int r2, int c1, int c2, int* seed);

#endif // LCG_H
```

Листинг A.36 lcg.c

```c
#include <lcg.h>

double linear_congruential_gen(int* seed) {
  const int Q = MODULUS / MULTIPLIER;
  const int R = MODULUS % MULTIPLIER;
  int t = MULTIPLIER * (*seed % Q) - R * (*seed / Q);
  if (t > 0)
    *seed = t;
  else
    *seed = t + MODULUS;
  return (double) *seed / MODULUS;
}

void fill_vector_double(double* v, int len, int i1, int i2, int* seed) {
  if (i2 > len)
    i2 = len;
  for (int i = i1; i < i2; i++)
    v[i] = linear_congruential_gen(seed) * len;
}

void fill_matrix_double(double** m, int rows, int cols, int r1, int r2, int c1, int c2, int* seed) {
  if (r2 > rows)
    r2 = rows;
  for (int i = r1; i < r2; i++)
    fill_vector_double(m[i], cols, c1, c2, seed);
}
```

A.4 Работа с памятью и измерение времени выполнения программ

Листинг A.37 util.h

```c
#ifndef UTIL_H
#define UTIL_H

#include <sys/time.h>
#include <stddef.h>

#define elapsed_msecs(s, f) (1000.0 * ((f).tv_sec - (s).tv_sec) + 0.001 * ((f).tv_usec -
    ↪ (s).tv_usec))

double rtclock();

double mean_value(double* list, int len);
double var_value(double* list, int len, double mean);

double min_value(double* list, int len);
double max_value(double* list, int len);

void print_stats(double* list, int len, const char* hint, const char* unit);

void* allocate_vector(int len, size_t elem_size, const char* name);
void** allocate_matrix(int rows, int cols, size_t elem_size, const char* name);

void free_matrix(void** m, int rows);

double* allocate_vector_double(int len, const char* name);
double** allocate_matrix_double(int rows, int cols, const char* name);

void free_matrix_double(double** m, int rows);

double compare_vectors_double(double* v1, double* v2, int len);
double compare_matrices_double(double** m1, double** m2, int rows, int cols);

void copy_matrix_double(double** src, double** dst, int r1, int r2, int c1, int c2);

void zero_vector_double(double* v, int len, int i1, int i2);
void zero_matrix_double(double** m, int rows, int cols, int r1, int r2, int c1, int c2);

#endif // UTIL_H
```

Листинг A.38 util.c

```c
#include <util.h>

#include <sys/mman.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <math.h>
#include <float.h>

double rtclock() {
  struct timeval tv;
  int stat = gettimeofday(&tv, 0);
```

```
  if (stat != 0)
    fprintf(stderr, "Error returned from gettimeofday: %d", stat);
  return tv.tv_sec + tv.tv_usec * 1.0e-6;
}

double mean_value(double* list, int len) {
  double sum = 0;
  for (int i = 0; i < len; i++)
    sum += list[i];
  return sum / len;
}

double var_value(double* list, int len, double mean) {
  double sum = 0;
  for (int i = 0; i < len; i++)
    sum += (list[i] - mean) * (list[i] - mean);
  return sqrt(sum / len);
}

double min_value(double* list, int len) {
  double r = DBL_MAX;
  for (int i = 0; i < len; i++)
    if (list[i] < r)
      r = list[i];
  return r;
}

double max_value(double* list, int len) {
  double r = DBL_MIN;
  for (int i = 0; i < len; i++)
    if (list[i] > r)
      r = list[i];
  return r;
}

void print_stats(double* list, int len, const char* hint, const char* unit) {
  double listMean = mean_value(list, len);
  printf("%sMean: %lf%s\n", hint, listMean, unit);
  double listVar = var_value(list, len, listMean);
  printf("%sSD: %lf%s\n", hint, listVar, unit);
  double listMin = min_value(list, len);
  printf("%sMin: %lf%s\n", hint, listMin, unit);
  double listMax = max_value(list, len);
  printf("%sMax: %lf%s\n", hint, listMax, unit);
}

void* allocate_vector(int len, size_t elem_size, const char* name) {
  if (len == 0)
    return NULL;
  size_t sz = elem_size * len;
  void* v = malloc(sz);
  assert(v != NULL);
  if (mlock(v, sz) != 0)
    fprintf(stderr, "Unable to lock buffer pages for %s\n", name);
  return v;
}

void** allocate_matrix(int rows, int cols, size_t elem_size, const char* name) {
  void** m = (void**) allocate_vector(rows, sizeof(void*), name);
  for (int row = 0; row < rows; row++) {
    char col_name_buf[50];
```

```
    sprintf(col_name_buf, "%s[%d]", name, row);
    m[row] = allocate_vector(cols, elem_size, col_name_buf);
  }
  return m;
}

void free_matrix(void** m, int rows) {
  for (int row = 0; row < rows; row++)
    free(m[row]);
  free(m);
}

double* allocate_vector_double(int len, const char* name) {
  return (double*) allocate_vector(len, sizeof(double), name);
}

double** allocate_matrix_double(int rows, int cols, const char* name) {
  return (double**) allocate_matrix(rows, cols, sizeof(double), name);
}

void free_matrix_double(double** m, int rows) {
  free_matrix((void**) m, rows);
}

double compare_vectors_double(double* v1, double* v2, int len) {
  double max_diff = 0;
  for (int i = 0; i < len; i++) {
    double diff = fabs(v1[i] / v2[i] - 1);
    if (diff > max_diff)
      max_diff = diff;
  }
  return max_diff;
}

double compare_matrices_double(double** m1, double** m2, int rows, int cols) {
  double max_diff = 0;
  for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
      double diff = fabs(m1[i][j] / m2[i][j] - 1);
      if (diff > max_diff)
        max_diff = diff;
    }
  }
  return max_diff;
}

void copy_matrix_double(double** src, double** dst, int r1, int r2, int c1, int c2) {
  for (int i = r1; i < r2; i++)
    for (int j = c1; j < c2; j++)
      dst[i][j] = src[i][j];
}

void zero_vector_double(double* v, int len, int i1, int i2) {
  if (i2 > len)
    i2 = len;
  for (int i = i1; i < i2; i++)
    v[i] = 0;
}

void zero_matrix_double(double** m, int rows, int cols, int r1, int r2, int c1, int c2) {
  if (r2 > rows)
```

```
    r2 = rows;
  for (int i = r1; i < r2; i++)
    zero_vector_double(m[i], cols, c1, c2);
}
```

A.5 Макросы cloog

Листинг A.39 umacros.h

```
#ifndef UMACROS_H
#define UMACROS_H

#include <math.h>

#ifndef ceild
#define ceild(n,d) ((int) ceil(((double)(n))/((double)(d))))
#endif
#ifndef floord
#define floord(n,d) ((int) floor(((double)(n))/((double)(d))))
#endif
#ifndef max
#define max(x,y)  ((x) > (y)? (x) : (y))
#endif
#ifndef min
#define min(x,y)  ((x) < (y)? (x) : (y))
#endif

#ifdef TIME
#define IF_TIME(foo) foo;
#else
#define IF_TIME(foo)
#endif

#endif // UMACROS_H
```

Приложение Б

Распараллеливание программы lu на языке C

Б.1 Описание программы

LU-разложение квадратной матрицы A сводится к представлению ее в виде произведения двух множителей — нижней треугольной матрицы L и верхней треугольной матрицы U: $A = LU$. Пусть матрицы A, L, U имеют размеры $N \times N$, тогда L и U вычисляются следующим образом:

```
for (int i = 0; i < N; i++) {
  for (int j = 0; j < N; j++) {
    U(i,j) ← 0;
    L(i,j) ← 0;
  }
  L(i,i) ← 1;
}
for (int i = 0; i < N; i++) {
  for (int j = 0; j < N; j++) {
    if (i <= j) U(i,j) ← A(i,j) − ∑_{k=0}^{i−1} L(i,k)U(k,j);
    if (i > j) L(i,j) ← (A(i,j) − ∑_{k=0}^{j−1} L(i,k)U(k,j))/U(j,j);
  }
}
```

Листинг Б.1 LU-разложение (последовательный вариант) на языке C

```
1  void lu_vanilla(int N, double** A) {
2  #pragma scop
3    for (int k = 0; k < N; k++) {
4      for (int l = k + 1; l < N; l++)
5        A[l][k] /= A[k][k]; //S0
6      for (int i = k + 1; i < N; i++)
7        for (int j = k + 1; j < N; j++)
8          A[i][j] -= A[i][k] * A[k][j]; //S1
9    }
10 #pragma endscop
11 }
```

На рисунке Б.1 проиллюстрирован обобщенный граф зависимостей фрагмента функции `lu_vanilla`. Ребра аннотированы сведениями об информационных зависимостях $u \to v$: операция доступа к памяти u должна предшествовать

v. Зависимости R0, R2, R4, R6, R8, R9 имеют тип «чтение после записи», зависимости R1, R5 имеют тип «запись после чтения», зависимости R3, R7 имеют тип «запись после записи».

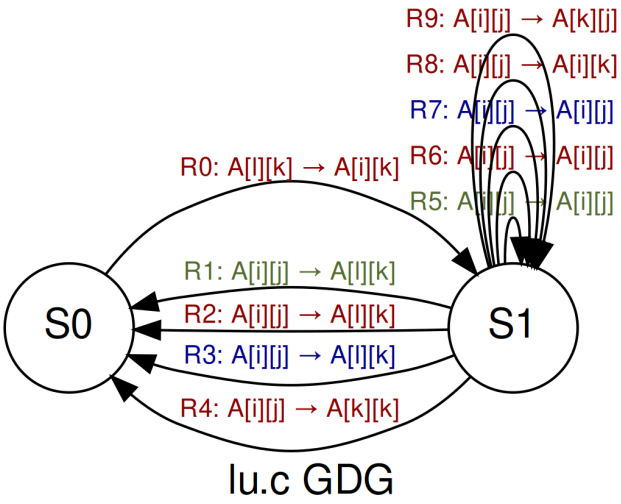


Рисунок Б.1 — Обобщенный граф зависимостей lu_vanilla

Б.2 Журнал трансляции ilpy

Листинг Б.2 Журнал трансляции ilpy при нахождении аффинных отображений для lu

```
#Расписание вычислений

R9 (weight 318549): L=2
R8 (weight 318549): L=2
R7 (weight 318549): L=2
R6 (weight 318549): L=2
R5 (weight 318549): L=2
R4 (weight 4851): L=1
R3 (weight 4851): L=1
R2 (weight 4851): L=1
R1 (weight 4851): L=1
R0 (weight 328350): L=1
zero: 0, constant: 10, affine: 0, total: 10
C = 3533244.000000

#Размещение вычислений

# component 1 with linear independence ENABLED and FCO property ENABLED
R9 (weight 318549): L=N
R8 (weight 318549): L=0
R7 (weight 318549): L=0
R6 (weight 318549): L=0
R5 (weight 318549): L=0
R4 (weight 4851): L=N
```

```
R3 (weight 4851): L=0
R2 (weight 4851): L=0
R1 (weight 4851): L=0
R0 (weight 328350): L=0
zero: 8, constant: 0, affine: 2, total: 10
C = 32340000.000000
```

Листинг Б.3 Журнал трансляции ilpy при нахождении размещения вычислений и данных для lu

```
# component 1 with linear independence ENABLED
    and FCO property ENABLED
S0, A2 (READ A[k][k], weight 4950): L=N
S0, A1 (WRITE A[l][k], weight 4950): L=0
S0, A0 (READ A[l][k], weight 4950): L=0
S1, A3 (READ A[k][j], weight 328350): L=N
S1, A2 (READ A[i][k], weight 328350): L=0
S1, A1 (WRITE A[i][j], weight 328350): L=0
S1, A0 (READ A[i][j], weight 328350): L=0
zero: 5, constant: 0, affine: 2, total: 7
C = 33330000.000000
```

Б.3 Результат работы ilpy

Листинг Б.4 lu (параллельный вариант ilpy без директив OpenMP, синхронный параллелизм) на языке C

```
if (N >= 2) {
 for (t0=0;t0<=2*N-3;t0++) {
  for (ilpp=ceild(t0-1,2);ilpp<=N-2;ilpp++) {
   if (t0%2 == 0) {
    A[ilpp+1][t0/2] /= A[t0/2][t0/2];
   }
   if ((t0+1)%2 == 0) {
    for (k=ceild(t0+1,2);k<=N-1;k++) {
     A[ilpp+1][k] -= A[ilpp+1][(t0-1)/2] * A[(t0-1)/2][k];
    }
   }
  }
 }
}
```

Листинг Б.5 lu_ilp_sync (параллельный вариант ilpy с директивами OpenMP, синхронный параллелизм) на языке C

```
void lu_ilp_sync(int N, double** A) {
  #pragma omp parallel
  {
    if (N >= 2) {
      for (int t0=0;t0<=2*N-3;t0++) {
        #pragma omp for
        for (int ilpp=ceild(t0-1,2);ilpp<=N-2;ilpp++) {
```

```
          if (t0%2 == 0) {
            A[ilpp+1][t0/2] /= A[t0/2][t0/2];
          }
          if ((t0+1)%2 == 0) {
            for (int k=ceild(t0+1,2);k<=N-1;k++) {
              A[ilpp+1][k] -= A[ilpp+1][(t0-1)/2] * A[(t0-1)/2][k];
            }
          }
        }
      }
    }
  }
}
```

Листинг Б.6 lu (параллельный вариант ilpy без конструкций MPI, синхронный параллелизм) на языке C

```
if (N >= 2) {
 for (t0=0;t0<=2*N-3;t0++) {
  for (ilpp=ceild(t0+1,2);ilpp<=N-1;ilpp++) {
   if (t0%2 == 0) {
    A[ilpp][t0/2] /= A[t0/2][t0/2];
   }
   if ((t0+1)%2 == 0) {
    for (k=ceild(t0+1,2);k<=N-1;k++) {
     A[ilpp][k] -= A[ilpp][(t0-1)/2] * A[(t0-1)/2][k];
    }
   }
  }
 }
}
```

Листинг Б.7 lu_ilp_arrays (параллельный вариант ilpy с конструкциями MPI, синхронный параллелизм) на языке C

```
// arrays placement

#define T_base 1000000

#define eta_A(i0,i1) (i0)
#define T_A (1 * T_base)

int A_chunk_size;
int A_actual_chunk_size;

// mpi routine

#define eta_A_t0_even(i1) eta_A(t0/2,i1)
#define eta_A_t0_odd(i1) eta_A((t0-1)/2,i1)

void lu_ilp_arrays(int N, double** A) {
  if (N >= 2) {
    for (int t0=0;t0<=2*N-3;t0++) {
      // A[t0/2][t0/2] in A[ilpp][t0/2] /= A[t0/2][t0/2]
      if (t0%2 == 0) {
        line_Q_read_vp(0, t0/2, ceild(t0+1,2), N-1, A[t0/2], eta_A_t0_even, T_A + t0/2);
      }
      //
      // A[(t0-1)/2][k] in A[ilpp][k] -= A[ilpp][(t0-1)/2] * A[(t0-1)/2][k]
```

```
      if ((t0+1)%2 == 0) {
        line_Q_read_send(ceild(t0+1,2), clamp(lr, ceild(t0+1,2), N-1), N-1, clamp(ur, ceild(t0+1,2),
    ↪ N-1), A[(t0-1)/2],
            eta_A_t0_odd, T_A + (t0-1)/2);
        line_Q_read_receive(ceild(t0+1,2), clamp(lR, ceild(t0+1,2), N-1), N-1, clamp(uR,
    ↪ ceild(t0+1,2), N-1), A[(t0-1)/2],
            eta_A_t0_odd, T_A + (t0-1)/2);
      }
      //
      for (int ilpp=max(lR,ceild(t0+1,2));ilpp<=min(uR,N-1);ilpp++) {
        if (t0%2 == 0) {
          A[ilpp][t0/2] /= A[t0/2][t0/2];
        }
        if ((t0+1)%2 == 0) {
          for (int k=ceild(t0+1,2);k<=N-1;k++) {
            A[ilpp][k] -= A[ilpp][(t0-1)/2] * A[(t0-1)/2][k];
          }
        }
      }
      MPI_Barrier_time(MPI_COMM_WORLD);
    }
    collect_matrix_rows_double(A, N, N, A_chunk_size);
  }
}
```

Б.4 Преобразования pluto

Листинг Б.8 Преобразования pluto в формате cloog для lu_pluto

```
# Number of scattering functions
2

# T(S1)
3 8
0 1 0 0 -1 -1 0 0
0 0 1 0 0 -1 0 0
0 0 0 1 -1 0 0 0

# T(S2)
3 9
0 1 0 0 -1 -1 0 0 0
0 0 1 0 0 -1 0 0 0
0 0 0 1 0 0 -1 0 0
```

Листинг Б.9 lu (параллельный вариант pluto с директивами OpenMP, синхронный параллелизм) на языке C

```
 int t1, t2, t3;
 int lb, ub, lbd, ubd, lb2, ub2;
 register int lbv, ubv;
/* Start of CLooG code */
```

```
if (N >= 2) {
 for (t1=1;t1<=2*N-3;t1++) {
  {/* extra braces to avoid redefination of lbp*/
int lbp=ceild(t1+1,2);
  int ubp=min(t1,N-1);
#pragma omp parallel for private(lbv,ubv,t3)
  for (t2=lbp;t2<=ubp;t2++) {
   A[t2][(t1-t2)] /= A[(t1-t2)][(t1-t2)];;
   for (t3=t1-t2+1;t3<=N-1;t3++) {
    A[t2][t3] -= A[t2][(t1-t2)] * A[(t1-t2)][t3];;
   }
  }
}/*end of omp parallel loop */
 }
}
/* End of CLooG code */
```

Листинг Б.10 lu_pluto (обработанный вариант pluto с директивами OpenMP, синхронный параллелизм) на языке C

```
void lu_pluto(int N, double** A) {
  #pragma omp parallel
  {
    /* Start of CLooG code */
    if (N >= 2) {
      for (int t1=1;t1<=2*N-3;t1++) {
        #pragma omp for
        for (int t2=ceild(t1+1,2);t2<=min(t1,N-1);t2++) {
          A[t2][(t1-t2)] /= A[(t1-t2)][(t1-t2)];
          for (int t3=t1-t2+1;t3<=N-1;t3++) {
            A[t2][t3] -= A[t2][(t1-t2)] * A[(t1-t2)][t3];
          }
        }
      }
    }
    /* End of CLooG code */
  }
}
```

Листинг Б.11 Преобразования pluto в формате cloog для lu_pluto_mpi

```
# Number of scattering functions
5

# T(S1)
4 10
0 1 0 0 0 -1 -1 0 0 0
0 0 1 0 0 0 0 0 0 0
0 0 0 1 0 0 -1 0 0 0
0 0 0 0 1 -1 0 0 0 0

# T(S2)
4 11
0 1 0 0 0 -1 -1 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0
0 0 0 1 0 0 -1 0 0 0 0
0 0 0 0 1 0 0 -1 0 0 0
```

Листинг Б.12 lu_pluto_mpi (параллельный вариант pluto с конструкциями MPI, синхронный параллелизм) на языке C

```
/* Start of CLooG code */
if (N >= 2) {
 for (t1=1;t1<=2*N-3;t1++) {
  IF_TIME(t_comp_start = rtclock());
_lb_dist=ceild(t1+1,2);
  _ub_dist=min(t1,N-1);
  polyrt_loop_dist(_lb_dist, _ub_dist, nprocs,
    ↪ my_rank, &lbd_t3, &ubd_t3);
  for (t3=lbd_t3;t3<=ubd_t3;t3++) {
   A[t3][(t1-t3)] /= A[(t1-t3)][(t1-t3)];;
   lbv=t1-t3+1;
   ubv=N-1;
#pragma ivdep
#pragma vector always
   for (t4=lbv;t4<=ubv;t4++) {
    A[t3][t4] -= A[t3][(t1-t3)] * A[(t1-t3)][t4];;
   }
  }
IF_TIME(t_comp += rtclock() - t_comp_start);
  IF_TIME(t_pack_start = rtclock());
_lb_dist=ceild(t1+1,2);
  _ub_dist=min(t1,N-1);
  polyrt_loop_dist(_lb_dist, _ub_dist, nprocs,
    ↪ my_rank, &lbd_t3, &ubd_t3);
  for (t3=lbd_t3;t3<=ubd_t3;t3++) {
   for (__p=0; __p<nprocs; __p++) { receiver_list[__p]
    ↪ = 0; } sigma_A_1_0(t1,t3,N, my_rank, nprocs,
    ↪ receiver_list); for (__p=0; __p<nprocs; __p++) {
    ↪ if (receiver_list[__p] != 0) { send_counts_A[__p]
    ↪ = pack_A_1_0(t1,t3, send_buf_A[__p], send_counts_A[__p]);
    ↪ } }for (__p=0; __p<nprocs; __p++) { receiver_list[__p]
    ↪ = 0; } sigma_A_2_0(t1,t3,N, my_rank, nprocs, receiver_list);
    ↪ for (__p=0; __p<nprocs; __p++) { if (receiver_list[__p]
    ↪ != 0) { send_counts_A[__p] = pack_A_2_0(t1,t3, send_buf_A[__p],
    ↪ send_counts_A[__p]); } }for (__p=0; __p<nprocs; __p++)
    ↪ { receiver_list[__p] = 0; } sigma_A_3_0(t1,t3,N,
    ↪ my_rank, nprocs, receiver_list); for (__p=0; __p<nprocs;
    ↪ __p++) { if (receiver_list[__p] != 0) { send_counts_A[__p]
    ↪ = pack_A_3_0(t1,t3, send_buf_A[__p], send_counts_A[__p]);
    ↪ } };
  }
IF_TIME(t_pack += rtclock() - t_pack_start);
  IF_TIME(t_comm_start = rtclock());
    ↪ MPI_Alltoall(send_counts_A, 1, MPI_INT,
    ↪ recv_counts_A, 1, MPI_INT, MPI_COMM_WORLD); req_count=0;
    ↪ for (__p=0; __p<nprocs; __p++) { if (send_counts_A[__p]
    ↪ >= 1) {IF_TIME(__total_count += send_counts_A[__p]);
    ↪ MPI_Isend(send_buf_A[__p], send_counts_A[__p], MPI_DOUBLE,
    ↪ __p, 123, MPI_COMM_WORLD, &reqs[req_count++]);}}for
    ↪ (__p=0; __p<nprocs; __p++) { if(recv_counts_A[__p]
    ↪ >= 1) { MPI_Irecv(recv_buf_A+displs_A[__p], recv_counts_A[__p],
    ↪ MPI_DOUBLE, __p, 123, MPI_COMM_WORLD, &reqs[req_count++]);}}
    ↪ MPI_Waitall(req_count, reqs, stats); for (__p=0;
    ↪ __p<nprocs; __p++) { send_counts_A[__p] = 0;}for
    ↪ (__p=0; __p<nprocs; __p++) { curr_displs_A[__p] =
    ↪ displs_A[__p]; }IF_TIME(t_comm += rtclock() - t_comm_start);;
  IF_TIME(t_unpack_start = rtclock());
```

```
for (t3=ceild(t1+1,2);t3<=min(t1,N-1);t3++) {
   proc = pi_0(t1,t3,N, nprocs); if ((my_rank != proc)
    ↪ && (recv_counts_A[proc] > 0)) { if
    ↪ (is_receiver_A_1_0(t1,t3,N,my_rank,nprocs) != 0)
    ↪ { curr_displs_A[proc] = unpack_A_1_0(t1,t3,recv_buf_A,curr_displs_A[proc]);
    ↪ }if (is_receiver_A_2_0(t1,t3,N,my_rank,nprocs) !=
    ↪ 0) { curr_displs_A[proc] = unpack_A_2_0(t1,t3,recv_buf_A,curr_displs_A[proc]);
    ↪ }if (is_receiver_A_3_0(t1,t3,N,my_rank,nprocs) !=
    ↪ 0) { curr_displs_A[proc] = unpack_A_3_0(t1,t3,recv_buf_A,curr_displs_A[proc]);
    ↪ }};
  }
IF_TIME(t_unpack += rtclock() - t_unpack_start);
 }
}
/* End of CLooG code */
```

Б.5 Статистика запусков lu (OpenMP)

Таблица 6 — Статистика запусков lu в вариантах vanilla, ilp_sync, pluto: затраченное время, мс

#Нитей	Стат.	vanilla	ilp_sync	pluto
1	μ	9180.168	9097.2473	11441.2498
	σ	34.81391	49.212905	85.30848
	Min	9163.699	9065.37	11395.586
	Max	9284.215	9215.257	11659.563
2	μ	–	5444.1325	7618.9598
	σ	–	0.818866	1.24945
	Min	–	5442.356	7616.915
	Max	–	5444.927	7621.287
4	μ	–	4386.0995	5862.9389
	σ	–	1.066748	0.31183
	Min	–	4384.842	5862.491
	Max	–	4387.862	5863.381
6	μ	–	4070.0436	5272.0599
	σ	–	0.56044	0.43195
	Min	–	4069.334	5271.372
	Max	–	4071.024	5272.798
8	μ	–	4042.2624	5054.7918
	σ	–	0.482905	24.582673
	Min	–	4041.341	5033.147
	Max	–	4043.171	5102.588

Б.6 Статистика запусков lu (MPI)

Таблица 7 — Статистика запусков lu_mpi в вариантах vanilla, ilp_arrays (_one, _two, _eq): затраченное время, мс

#Проц.	Стат.	vanilla	ilp_arrays_one	ilp_arrays_two	ilp_arrays_eq
1	μ	9068.1836	9086.51951	–	–
	σ	29.31987	66.804632	–	–
	Min	9048.5	9036.425	–	–
	Max	9224.05	9266.339	–	–
2	μ	–	6919.89142	5850.64746	–
	σ	–	9.233174	56.64428	–
	Min	–	6862.912	5798.739	–
	Max	–	6952.537	6015.749	–
4	μ	–	5121.58567	3493.39263	2709.2206
	σ	–	8.766815	3.896619	14.675053
	Min	–	5115.503	3488.593	2699.191
	Max	–	5146.882	3524.405	2843.856
6	μ	–	4566.63432	2815.79131	1862.46222
	σ	–	3.831584	2.952064	20.8414
	Min	–	4563.166	2811.265	1841.063
	Max	–	4593.853	2835.934	2027.33
8	μ	–	4385.55366	2522.36798	1457.71504
	σ	–	3.680072	3.810486	15.044019
	Min	–	4381.248	2517.24	1452.931
	Max	–	4413.048	2549.5	1605.452

Таблица 8 — Статистика запусков lu_mpi в вариантах vanilla, ilp_arrays (_one, _two, _eq): доля времени вычислений, %

#Проц.	Стат.	vanilla	ilp_arrays_one	ilp_arrays_two	ilp_arrays_eq
1	μ	100.0	99.982884	–	–
	σ	0.0	0.000402	–	–
	Min	100.0	99.981958	–	–
	Max	100.0	99.984066	–	–
2	μ	–	75.952598	71.159345	–
	σ	–	0.082119	0.159207	–
	Min	–	75.556156	70.882133	–
	Max	–	76.178158	71.5534	–
4	μ	–	74.535041	65.946124	65.143304
	σ	–	0.152707	0.027969	0.275641
	Min	–	74.112025	65.898035	64.564014
	Max	–	74.638309	66.08592	66.585762
6	μ	–	75.599081	64.122258	62.331644
	σ	–	0.049682	0.09332	0.229168
	Min	–	75.112106	63.956361	61.900598
	Max	–	75.618575	64.350917	63.451077
8	μ	–	76.465627	63.021588	59.992167
	σ	–	0.059589	0.073307	0.149646
	Min	–	75.878261	62.417263	59.68731
	Max	–	76.488596	63.091468	61.220013

Таблица 9 — Статистика запусков lu_mpi в вариантах pluto (_one, _two, _eq): затраченное время, мс

#Проц.	Стат.	pluto_one	pluto_two	pluto_eq
1	μ	11851.435711	–	–
	σ	127.414622	–	–
	Min	11685.008049	–	–
	Max	12098.301888	–	–
2	μ	7904.419315	5726.733344	–
	σ	58.486391	123.329345	–
	Min	7855.074167	5542.564154	–
	Max	8062.641144	5958.375216	–
4	μ	6258.013005	3369.854276	2952.829742
	σ	9.714397	59.798798	37.939836
	Min	6249.991179	3296.844959	2893.662214
	Max	6345.749855	3501.585007	3055.390835
6	μ	5678.486779	2683.789828	2121.079462
	σ	28.719149	42.182586	45.990887
	Min	5649.412155	2634.059906	2057.790041
	Max	5736.874104	2752.382994	2223.052979
8	μ	5562.213118	2327.870889	1720.373044
	σ	17.213052	31.230661	65.654196
	Min	5548.817873	2288.924932	1658.706903
	Max	5625.179052	2386.482	1837.602139

Таблица 10 — Статистика запусков lu_mpi в вариантах pluto (_one, _two, _eq): доля времени вычислений, %

#Проц.	Стат.	pluto_one	pluto_two	pluto_eq
1	μ	97.122672	–	–
	σ	0.037972	–	–
	Min	96.984719	–	–
	Max	97.160292	–	–
2	μ	87.265641	81.397102	–
	σ	0.400208	0.611618	–
	Min	86.641356	80.20701	–
	Max	87.571492	82.35449	–
4	μ	84.103131	73.145808	73.395852
	σ	0.040274	0.461507	0.90081
	Min	83.749642	71.938689	71.420409
	Max	84.14367	73.998958	75.033461
6	μ	84.55416	67.182557	67.726127
	σ	0.379489	0.364559	1.435977
	Min	83.873446	66.232833	64.331145
	Max	84.864328	67.657177	70.420353
8	μ	83.773514	62.715402	62.177937
	σ	0.140621	0.235329	2.315401
	Min	83.272124	62.295389	58.304422
	Max	83.858502	63.001449	64.753151

Таблица 11 — Разнородная нагрузка в запусках lu_mpi

Запуск	Затраченное время, мс			Доля затраченного времени, %		
vanilla: 1=1x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	9062.38	0.00	0.00	100.00	0.00	0.00
ilp_arrays_one: 1=1x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	9086.18	0.53	0.98	99.98	0.01	0.01

Продолжение таблицы 11

Запуск	Затраченное время, мс			Доля затраченного времени, %		
ilp_arrays_one: 2=1x2	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	6920.68	14.30	3266.45	52.59	0.21	47.20
Process 1	6920.68	41.10	5.47	99.33	0.59	0.08
ilp_arrays_two: 2=2x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	5870.56	25.99	3309.39	43.18	0.44	56.37
Process 1	5870.56	40.48	18.51	99.00	0.69	0.32
ilp_arrays_one: 4=1x4	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	5121.17	21.70	3392.70	33.33	0.42	66.25
Process 1	5121.17	44.38	1224.12	75.23	0.87	23.90
Process 2	5121.17	55.27	387.81	91.35	1.08	7.57
Process 3	5121.17	57.84	36.49	98.16	1.13	0.71
ilp_arrays_two: 4=2x2	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	3493.04	33.77	2623.30	23.93	0.97	75.10
Process 1	3493.04	39.35	1527.72	55.14	1.13	43.74
Process 2	3493.03	48.55	398.59	87.20	1.39	11.41
Process 3	3493.03	58.86	23.85	97.63	1.69	0.68
ilp_arrays_eq: 4=4x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	2707.80	40.10	2062.58	22.35	1.48	76.17
Process 1	2707.81	49.16	1095.20	57.74	1.82	40.45
Process 2	2707.80	64.44	368.67	84.01	2.38	13.62
Process 3	2707.81	76.95	20.23	96.41	2.84	0.75
ilp_arrays_one: 6=1x6	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	4566.06	25.75	3416.30	24.62	0.56	74.82
Process 1	4566.06	41.76	1700.25	61.85	0.91	37.24
Process 2	4566.06	53.03	739.35	82.65	1.16	16.19
Process 3	4566.06	59.18	347.57	91.09	1.30	7.61
Process 4	4566.06	62.36	148.96	95.37	1.37	3.26
Process 5	4566.06	62.90	25.73	98.06	1.38	0.56
ilp_arrays_two: 6=2x3	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	2815.50	37.04	2272.50	17.97	1.32	80.71
Process 1	2815.50	41.55	1660.09	39.56	1.48	58.96
Process 2	2815.50	43.99	1243.13	54.28	1.56	44.15
Process 3	2815.49	50.85	368.05	85.12	1.81	13.07
Process 4	2815.49	63.48	149.24	92.44	2.25	5.30
Process 5	2815.49	70.40	21.87	96.72	2.50	0.78
ilp_arrays_eq: 6=6x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	1862.25	49.00	1539.47	14.70	2.63	82.67
Process 1	1862.25	57.10	1055.32	40.26	3.07	56.67
Process 2	1862.25	64.30	645.79	61.87	3.45	34.68
Process 3	1862.25	76.32	359.91	76.57	4.10	19.33
Process 4	1862.25	84.24	153.94	87.21	4.52	8.27
Process 5	1862.25	97.87	32.46	93.00	5.26	1.74
ilp_arrays_one: 8=1x8	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	4385.35	28.76	3503.77	19.45	0.66	79.90
Process 1	4385.35	42.22	2071.78	51.79	0.96	47.24
Process 2	4385.35	53.54	1060.86	74.59	1.22	24.19
Process 3	4385.35	61.59	539.39	86.30	1.40	12.30
Process 4	4385.35	65.41	328.62	91.01	1.49	7.49
Process 5	4385.35	68.58	171.75	94.52	1.56	3.92
Process 6	4385.35	69.48	79.89	96.59	1.58	1.82
Process 7	4385.35	70.23	36.14	97.57	1.60	0.82
ilp_arrays_two: 8=2x4	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	2522.07	41.75	2109.48	14.70	1.66	83.64
Process 1	2522.07	45.45	1683.90	31.43	1.80	66.77
Process 2	2522.07	47.65	1398.94	42.64	1.89	55.47
Process 3	2522.07	49.45	1175.88	51.42	1.96	46.62
Process 4	2522.07	55.57	337.57	84.41	2.20	13.38

Продолжение таблицы 11

Запуск	Затраченное время, мс			Доля затраченного времени, %		
Process 5	2522.07	68.50	179.17	90.18	2.72	7.10
Process 6	2522.07	77.95	77.78	93.83	3.09	3.08
Process 7	2522.07	84.53	30.92	95.42	3.35	1.23
ilp_arrays_eq: 8=8x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	1456.24	51.21	1248.73	10.73	3.52	85.75
Process 1	1456.25	55.57	958.79	30.34	3.82	65.84
Process 2	1456.24	64.31	684.17	48.60	4.42	46.98
Process 3	1456.25	71.20	492.85	61.27	4.89	33.84
Process 4	1456.24	81.13	315.69	72.75	5.57	21.68
Process 5	1456.25	92.73	197.91	80.04	6.37	13.59
Process 6	1456.24	109.24	81.87	86.88	7.50	5.62
Process 7	1456.25	121.14	23.05	90.10	8.32	1.58
pluto_one: 1=1x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	11838.96	349.65	0.00	97.05	2.95	0.00
pluto_one: 2=1x2	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	7906.79	1678.52	0.00	78.77	21.23	0.00
Process 1	7906.79	351.22	0.00	95.56	4.44	0.00
pluto_two: 2=2x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	5727.29	1618.28	0.00	71.74	28.26	0.00
Process 1	5727.29	508.69	0.00	91.12	8.88	0.00
pluto_one: 4=1x4	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	6257.07	1748.28	0.00	72.06	27.94	0.00
Process 1	6257.07	1152.38	0.00	81.58	18.42	0.00
Process 2	6257.07	715.12	0.00	88.57	11.43	0.00
Process 3	6257.07	356.40	0.00	94.30	5.70	0.00
pluto_two: 4=2x2	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	3370.19	1359.57	0.00	59.66	40.34	0.00
Process 1	3370.19	1116.35	0.00	66.88	33.12	0.00
Process 2	3370.18	624.83	0.00	81.46	18.54	0.00
Process 3	3370.18	404.39	0.00	88.00	12.00	0.00
pluto_eq: 4=4x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	2954.26	1251.56	0.00	57.64	42.36	0.00
Process 1	2954.26	965.01	0.00	67.33	32.67	0.00
Process 2	2954.26	537.85	0.00	81.79	18.21	0.00
Process 3	2954.26	443.59	0.00	84.98	15.02	0.00
pluto_one: 6=1x6	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	5686.32	1616.71	0.00	71.57	28.43	0.00
Process 1	5686.32	1210.18	0.00	78.72	21.28	0.00
Process 2	5686.32	910.90	0.00	83.98	16.02	0.00
Process 3	5686.32	641.27	0.00	88.72	11.28	0.00
Process 4	5686.32	470.85	0.00	91.72	8.28	0.00
Process 5	5686.32	371.65	0.00	93.46	6.54	0.00
pluto_two: 6=2x3	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	2683.38	1368.99	0.00	48.98	51.02	0.00
Process 1	2683.38	1196.76	0.00	55.40	44.60	0.00
Process 2	2683.38	1069.42	0.00	60.15	39.85	0.00
Process 3	2683.38	691.78	0.00	74.22	25.78	0.00
Process 4	2683.38	532.98	0.00	80.14	19.86	0.00
Process 5	2683.38	407.15	0.00	84.83	15.17	0.00
pluto_eq: 6=6x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	2121.25	1011.94	0.00	52.29	47.71	0.00
Process 1	2121.25	902.06	0.00	57.48	42.52	0.00
Process 2	2121.26	787.02	0.00	62.90	37.10	0.00
Process 3	2121.25	476.31	0.00	77.55	22.45	0.00
Process 4	2121.26	540.43	0.00	74.52	25.48	0.00
Process 5	2121.25	427.44	0.00	79.85	20.15	0.00
pluto_one: 8=1x8	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация

Продолжение таблицы 11

Запуск	Затраченное время, мс			Доля затраченного времени, %		
Process 0	5560.81	1690.02	0.00	69.61	30.39	0.00
Process 1	5560.81	1352.10	0.00	75.69	24.31	0.00
Process 2	5560.81	1099.83	0.00	80.22	19.78	0.00
Process 3	5560.81	878.86	0.00	84.20	15.80	0.00
Process 4	5560.81	733.56	0.00	86.81	13.19	0.00
Process 5	5560.81	585.52	0.00	89.47	10.53	0.00
Process 6	5560.81	474.74	0.00	91.46	8.54	0.00
Process 7	5560.81	374.43	0.00	93.27	6.73	0.00
pluto_two: 8=2x4	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	2328.86	1293.92	0.00	44.44	55.56	0.00
Process 1	2328.86	1214.83	0.00	47.84	52.16	0.00
Process 2	2328.86	1137.15	0.00	51.17	48.83	0.00
Process 3	2328.86	1061.84	0.00	54.41	45.59	0.00
Process 4	2328.87	679.50	0.00	70.82	29.18	0.00
Process 5	2328.87	584.18	0.00	74.92	25.08	0.00
Process 6	2328.87	504.16	0.00	78.35	21.65	0.00
Process 7	2328.87	426.82	0.00	81.67	18.33	0.00
pluto_eq: 8=8x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	1732.14	804.63	0.00	53.55	46.45	0.00
Process 1	1732.15	849.97	0.00	50.93	49.07	0.00
Process 2	1732.14	784.58	0.00	54.70	45.30	0.00
Process 3	1732.14	721.61	0.00	58.34	41.66	0.00
Process 4	1732.13	657.19	0.00	62.06	37.94	0.00
Process 5	1732.14	579.65	0.00	66.54	33.46	0.00
Process 6	1732.14	532.71	0.00	69.25	30.75	0.00
Process 7	1732.14	453.48	0.00	73.82	26.18	0.00

Приложение В

Распараллеливание программы atax на языке C

В.1 Описание программы

Матричное произведение atax подразумевает вычисление $y = A^T(Ax)$, где A — матрица размера $M \times N$, x и y — векторы длины N.

Листинг В.1 Произведение atax_p (последовательный вариант) на языке C

```
#pragma scop
  for (int i = 0; i < N; i++)
    y[i] = 0; //S0
  for (int i = 0; i < M; i++)
  {
    tmp[i] = 0; //S1
    for (int j = 0; j < N; j++)
      tmp[i] = tmp[i] + A[i][j] * x[j]; //S2
    for (int j = 0; j < N; j++)
      y[j] = y[j] + A[i][j] * tmp[i]; //S3
  }
#pragma endscop
```

Листинг В.2 Произведение atax (последовательный вариант) на языке C

```
#pragma scop
  for (int i = 0; i < N; i++)
    y[i] = 0; //S0
  for (int i = 0; i < M; i++)
  {
    tmp = 0; //S1
    for (int j = 0; j < N; j++)
      tmp = tmp + A[i][j] * x[j]; //S2
    for (int j = 0; j < N; j++)
      y[j] = y[j] + A[i][j] * tmp; //S3
  }
#pragma endscop
```

На рисунках В.1 и В.2 проиллюстрирован обобщенный граф зависимостей для каждого варианта программы. Для программы `atax_p` зависимости R1, R3, R4, R6, R8, R10 имеют тип «чтение после записи», зависимости R7, R11 имеют тип «запись после чтения», зависимости R0, R2, R5, R9 имеют тип «запись после

записи». Для программы `atax` зависимости R1, R4, R7, R11, R12, R15, R18 имеют тип «чтение после записи», зависимости R6, R8, R13, R14, R16, R19 имеют тип «запись после чтения», зависимости R0, R2, R3, R5, R9, R10, R17 имеют тип «запись после записи».

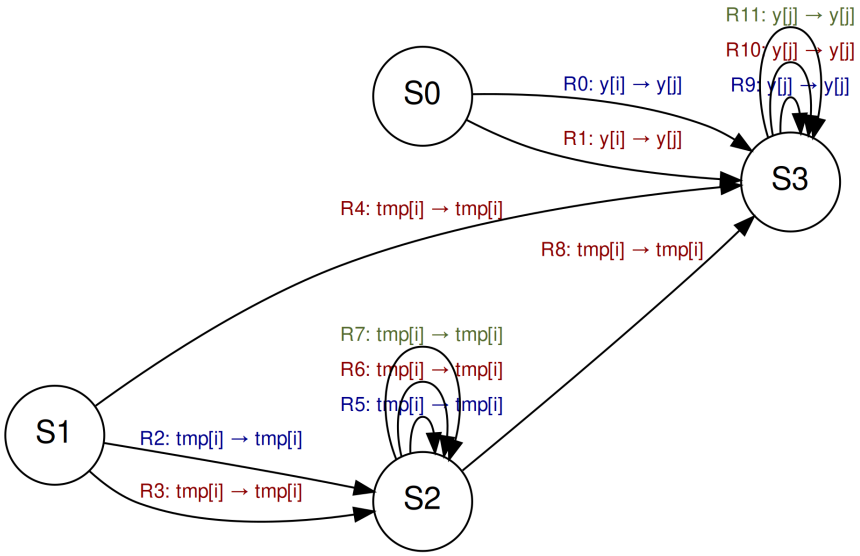


Рисунок B.1 — Обобщенный граф зависимостей atax_p

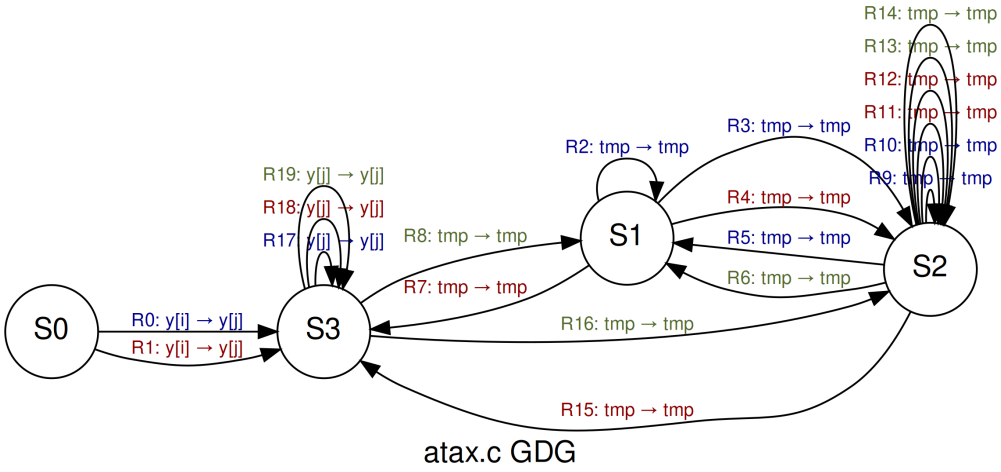


Рисунок B.2 — Обобщенный граф зависимостей atax

B.2 Журнал трансляции ilpy

Листинг B.3 Журнал трансляции ilpy при нахождении аффинных отображений для atax_p

```
#Расписание вычислений
```

```
R11 (weight 9900): L=1
R10 (weight 9900): L=1
R9 (weight 9900): L=1
R8 (weight 1000000): L=N
R7 (weight 9900): L=1
R6 (weight 9900): L=1
R5 (weight 9900): L=1
R4 (weight 10000): L=N+1
R3 (weight 10000): L=N
R2 (weight 10000): L=N
R1 (weight 10000): L=M
R0 (weight 10000): L=M
zero: 0, constant: 6, affine: 6, total: 12
C = 105069400.000000
```

```
#Размещение вычислений
```

```
# component 1 with linear independence ENABLED and FCO property ENABLED
R11 (weight 9900): L=1
R10 (weight 9900): L=1
R9 (weight 9900): L=1
R8 (weight 1000000): L=N
R7 (weight 9900): L=0
R6 (weight 9900): L=0
R5 (weight 9900): L=0
R4 (weight 10000): L=N
R3 (weight 10000): L=0
R2 (weight 10000): L=0
R1 (weight 10000): L=M
R0 (weight 10000): L=M
zero: 5, constant: 3, affine: 4, total: 12
C = 103029700.000000
```

Листинг В.4 Журнал трансляции ilpy при нахождении аффинных отображений для atax

```
#Расписание вычислений, первый компонент
```

```
R19 (weight 9900): L=3
R18 (weight 9900): L=3
R17 (weight 9900): L=3
R16 (weight 990000): L=2
R15 (weight 1000000): L=1
R13 (weight 990000): L=3
R11 (weight 990000): L=3
R9 (weight 990000): L=3
R8 (weight 9900): L=1
R7 (weight 10000): L=2
R6 (weight 9900): L=2
R5 (weight 9900): L=2
R4 (weight 10000): L=1
R3 (weight 10000): L=1
R2 (weight 99): L=3
R1 (weight 10000): L=3M
R0 (weight 10000): L=3M
zero: 0, constant: 15, affine: 2, total: 17
C = 18068897.000000
```

```
#Расписание вычислений, второй компонент
```

```
R14 (weight 9900): L=1
R12 (weight 9900): L=1
R10 (weight 9900): L=1
zero: 0, constant: 3, affine: 0, total: 3
C = 29700.000000

#Размещение вычислений

# component 1 with linear independence ENABLED and FCO property DISABLED
R19 (weight 9900): L=1
R18 (weight 9900): L=1
R17 (weight 9900): L=1
R16 (weight 990000): L=N
R15 (weight 1000000): L=N
R14 (weight 9900): L=0
R13 (weight 990000): L=1
R12 (weight 9900): L=0
R11 (weight 990000): L=1
R10 (weight 9900): L=0
R9 (weight 990000): L=1
R8 (weight 9900): L=N
R7 (weight 10000): L=N
R6 (weight 9900): L=1
R5 (weight 9900): L=1
R4 (weight 10000): L=0
R3 (weight 10000): L=0
R2 (weight 99): L=1
R1 (weight 10000): L=M
R0 (weight 10000): L=M
zero: 5, constant: 9, affine: 6, total: 20
C = 206009599.000000
```

Листинг В.5 Журнал трансляции ilpy при нахождении аффинных отображений для atax_p (расписание на основе топологической сортировки)

```
#Расписание вычислений, первый компонент

R8 (weight 1000000): L=1
R4 (weight 10000): L=2
R3 (weight 10000): L=1
R2 (weight 10000): L=1
R1 (weight 10000): L=2
R0 (weight 10000): L=2
zero: 0, constant: 6, affine: 0, total: 6
C = 1080000.000000

#Расписание вычислений, второй компонент

R11 (weight 9900): L=1
R10 (weight 9900): L=1
R9 (weight 9900): L=1
R7 (weight 9900): L=1
R6 (weight 9900): L=1
R5 (weight 9900): L=1
zero: 0, constant: 6, affine: 0, total: 6
C = 59400.000000

#Размещение вычислений
```

```
# component 1 with linear independence ENABLED and FCO property ENABLED
R11 (weight 9900): L=1
R10 (weight 9900): L=1
R9 (weight 9900): L=1
R8 (weight 1000000): L=N
R7 (weight 9900): L=0
R6 (weight 9900): L=0
R5 (weight 9900): L=0
R4 (weight 10000): L=N
R3 (weight 10000): L=0
R2 (weight 10000): L=0
R1 (weight 10000): L=M
R0 (weight 10000): L=M
zero: 5, constant: 3, affine: 4, total: 12
C = 103029700.000000
```

Листинг В.6 Журнал трансляции ilpy при нахождении размещения вычислений и данных для atax_p

```
# component 1 with linear independence ENABLED and FCO property DISABLED
S0, A0 (WRITE y[i], weight 100): L=0
S1, A0 (WRITE tmp[i], weight 100): L=M
S2, A3 (READ x[j], weight 10000): L=0
S2, A2 (READ A[i][j], weight 10000): L=0
S2, A1 (READ tmp[i], weight 10000): L=N
S2, A0 (WRITE tmp[i], weight 10000): L=N
S3, A3 (READ tmp[i], weight 10000): L=N
S3, A2 (READ A[i][j], weight 10000): L=0
S3, A1 (READ y[j], weight 10000): L=0
S3, A0 (WRITE y[j], weight 10000): L=0
zero: 6, constant: 0, affine: 4, total: 10
C = 3010000.000000
```

Листинг В.7 Журнал трансляции ilpy при нахождении размещения вычислений и данных для atax_p (расписание на основе топологической сортировки)

```
# component 1 with linear independence ENABLED and FCO property DISABLED
S0, A0 (WRITE y[i], weight 100): L=0
S1, A0 (WRITE tmp[i], weight 100): L=0
S2, A3 (READ x[j], weight 10000): L=M
S2, A2 (READ A[i][j], weight 10000): L=N+M
S2, A1 (READ tmp[i], weight 10000): L=0
S2, A0 (WRITE tmp[i], weight 10000): L=0
S3, A3 (READ tmp[i], weight 10000): L=N+M
S3, A2 (READ A[i][j], weight 10000): L=0
S3, A1 (READ y[j], weight 10000): L=0
S3, A0 (WRITE y[j], weight 10000): L=0
zero: 7, constant: 0, affine: 3, total: 10
C = 5000000.000000
```

В.3 Результат работы ilpy

Листинг В.8 atax_p (параллельный вариант ilpy без директив OpenMP, синхронный параллелизм) на языке C

```
if ((M <= 0) && (N >= 1)) {
 for (ilpp=0;ilpp<=N-1;ilpp++) {
```

```
  y[-ilpp+N-1] = 0;
 }
}
if ((M >= 1) && (N == 0)) {
 for (ilpp=0;ilpp<=0;ilpp++) {
  tmp[0] = 0;
 }
}
if ((M >= 1) && (N >= 1)) {
 for (ilpp=0;ilpp<=0;ilpp++) {
  tmp[0] = 0;
 }
}
if (N <= 0) {
 for (t0=max(0,N+1);t0<=M-1;t0++) {
  for (ilpp=t0;ilpp<=t0;ilpp++) {
   tmp[t0] = 0;
  }
 }
}
for (t0=1;t0<=min(M-1,N-1);t0++) {
 for (ilpp=0;ilpp<=t0-1;ilpp++) {
  tmp[ilpp] = tmp[ilpp] + A[ilpp][t0-ilpp-1] * x[t0-ilpp-1];
 }
 for (ilpp=t0;ilpp<=t0;ilpp++) {
  tmp[t0] = 0;
 }
}
if (M >= 1) {
 for (t0=M;t0<=N-1;t0++) {
  for (ilpp=0;ilpp<=M-1;ilpp++) {
   tmp[ilpp] = tmp[ilpp] + A[ilpp][t0-ilpp-1] * x[t0-ilpp-1];
  }
 }
}
if ((N >= 1) && (N <= M-1)) {
 for (ilpp=0;ilpp<=N-1;ilpp++) {
  if (2*ilpp == N-1) {
   if ((N+1)%2 == 0) {
    y[(N-1)/2] = 0;
    tmp[(N-1)/2] = tmp[(N-1)/2] + A[(N-1)/2][(N-1)/2] * x[(N-1)/2];
   }
  }
  if (ilpp >= ceild(N,2)) {
   y[-ilpp+N-1] = 0;
  }
  if (ilpp <= floord(N-2,2)) {
   tmp[ilpp] = tmp[ilpp] + A[ilpp][-ilpp+N-1] * x[-ilpp+N-1];
  }
  if (ilpp >= ceild(N,2)) {
   tmp[ilpp] = tmp[ilpp] + A[ilpp][-ilpp+N-1] * x[-ilpp+N-1];
  }
  if (ilpp <= floord(N-2,2)) {
   y[-ilpp+N-1] = 0;
  }
 }
 for (ilpp=N;ilpp<=N;ilpp++) {
  tmp[N] = 0;
 }
}
if ((M >= 1) && (N >= M)) {
```

```
 for (ilpp=0;ilpp<=M-1;ilpp++) {
  if (2*ilpp == N-1) {
   if ((N+1)%2 == 0) {
    y[(N-1)/2] = 0;
    tmp[(N-1)/2] = tmp[(N-1)/2] + A[(N-1)/2][(N-1)/2] * x[(N-1)/2];
   }
  }
  if (ilpp >= ceild(N,2)) {
   y[-ilpp+N-1] = 0;
  }
  if (ilpp <= floord(N-2,2)) {
   tmp[ilpp] = tmp[ilpp] + A[ilpp][-ilpp+N-1] * x[-ilpp+N-1];
  }
  if (ilpp >= ceild(N,2)) {
   tmp[ilpp] = tmp[ilpp] + A[ilpp][-ilpp+N-1] * x[-ilpp+N-1];
  }
  if (ilpp <= floord(N-2,2)) {
   y[-ilpp+N-1] = 0;
  }
 }
 for (ilpp=M;ilpp<=N-1;ilpp++) {
  y[-ilpp+N-1] = 0;
 }
}
if (N >= 1) {
 for (t0=N+1;t0<=M-1;t0++) {
  for (ilpp=t0-N;ilpp<=t0-1;ilpp++) {
   y[t0-ilpp-1] = y[t0-ilpp-1] + A[t0-N-1][t0-ilpp-1] * tmp[t0-N-1];
   tmp[ilpp] = tmp[ilpp] + A[ilpp][t0-ilpp-1] * x[t0-ilpp-1];
  }
  for (ilpp=t0;ilpp<=t0;ilpp++) {
   tmp[t0] = 0;
  }
 }
}
for (t0=max(M,N+1);t0<=N+M-1;t0++) {
 for (ilpp=t0-N;ilpp<=M-1;ilpp++) {
  y[t0-ilpp-1] = y[t0-ilpp-1] + A[t0-N-1][t0-ilpp-1] * tmp[t0-N-1];
  tmp[ilpp] = tmp[ilpp] + A[ilpp][t0-ilpp-1] * x[t0-ilpp-1];
 }
 for (ilpp=M;ilpp<=t0-1;ilpp++) {
  y[t0-ilpp-1] = y[t0-ilpp-1] + A[t0-N-1][t0-ilpp-1] * tmp[t0-N-1];
 }
}
if ((M >= 1) && (N >= 1)) {
 for (ilpp=M;ilpp<=N+M-1;ilpp++) {
  y[-ilpp+N+M-1] = y[-ilpp+N+M-1] + A[M-1][-ilpp+N+M-1] * tmp[M-1];
 }
}
```

Листинг B.9 atax_p_ilp_sync (параллельный вариант ilpy с директивами OpenMP, синхронный параллелизм) на языке C

```
void atax_p_ilp_sync(int M, int N, double** A, double* x, double* y, double* tmp) {
  #pragma omp parallel
  {
    if ((M <= 0) && (N >= 1)) {
      #pragma omp barrier
      #pragma omp for
      for (int ilpp=0;ilpp<=N-1;ilpp++) {
```

```
      y[-ilpp+N-1] = 0;
    }
  }
  if ((M >= 1) && (N == 0)) {
    #pragma omp master
    for (int ilpp=0;ilpp<=0;ilpp++) {
      tmp[0] = 0;
    }
  }
  if ((M >= 1) && (N >= 1)) {
    #pragma omp master
    for (int ilpp=0;ilpp<=0;ilpp++) {
      tmp[0] = 0;
    }
  }
  if (N <= 0) {
    for (int t0=max(0,N+1);t0<=M-1;t0++) {
      #pragma omp master
      for (int ilpp=t0;ilpp<=t0;ilpp++) {
        tmp[t0] = 0;
      }
    }
  }
  for (int t0=1;t0<=min(M-1,N-1);t0++) {
    #pragma omp barrier
    #pragma omp for
    for (int ilpp=0;ilpp<=t0-1;ilpp++) {
      tmp[ilpp] = tmp[ilpp] + A[ilpp][t0-ilpp-1] * x[t0-ilpp-1];
    }
    #pragma omp master
    for (int ilpp=t0;ilpp<=t0;ilpp++) {
      tmp[t0] = 0;
    }
  }
  if (M >= 1) {
    for (int t0=M;t0<=N-1;t0++) {
      #pragma omp barrier
      #pragma omp for
      for (int ilpp=0;ilpp<=M-1;ilpp++) {
        tmp[ilpp] = tmp[ilpp] + A[ilpp][t0-ilpp-1] * x[t0-ilpp-1];
      }
    }
  }
  if ((N >= 1) && (N <= M-1)) {
    #pragma omp barrier
    #pragma omp for
    for (int ilpp=0;ilpp<=N-1;ilpp++) {
      if (2*ilpp == N-1) {
        if ((N+1)%2 == 0) {
          y[(N-1)/2] = 0;
          tmp[(N-1)/2] = tmp[(N-1)/2] + A[(N-1)/2][(N-1)/2] * x[(N-1)/2];
        }
      }
      if (ilpp >= ceild(N,2)) {
        y[-ilpp+N-1] = 0;
      }
      if (ilpp <= floord(N-2,2)) {
        tmp[ilpp] = tmp[ilpp] + A[ilpp][-ilpp+N-1] * x[-ilpp+N-1];
      }
      if (ilpp >= ceild(N,2)) {
        tmp[ilpp] = tmp[ilpp] + A[ilpp][-ilpp+N-1] * x[-ilpp+N-1];
```

```
      }
70    if (ilpp <= floord(N-2,2)) {
        y[-ilpp+N-1] = 0;
      }
    }
    #pragma omp master
75  for (int ilpp=N;ilpp<=N;ilpp++) {
      tmp[N] = 0;
    }
  }
  if ((M >= 1) && (N >= M)) {
80  #pragma omp barrier
    #pragma omp for
    for (int ilpp=0;ilpp<=M-1;ilpp++) {
      if (2*ilpp == N-1) {
        if ((N+1)%2 == 0) {
85        y[(N-1)/2] = 0;
          tmp[(N-1)/2] = tmp[(N-1)/2] + A[(N-1)/2][(N-1)/2] * x[(N-1)/2];
        }
      }
      if (ilpp >= ceild(N,2)) {
90      y[-ilpp+N-1] = 0;
      }
      if (ilpp <= floord(N-2,2)) {
        tmp[ilpp] = tmp[ilpp] + A[ilpp][-ilpp+N-1] * x[-ilpp+N-1];
      }
95    if (ilpp >= ceild(N,2)) {
        tmp[ilpp] = tmp[ilpp] + A[ilpp][-ilpp+N-1] * x[-ilpp+N-1];
      }
      if (ilpp <= floord(N-2,2)) {
        y[-ilpp+N-1] = 0;
100   }
    }
    #pragma omp barrier
    #pragma omp for
    for (int ilpp=M;ilpp<=N-1;ilpp++) {
105   y[-ilpp+N-1] = 0;
    }
  }
  if (N >= 1) {
    for (int t0=N+1;t0<=M-1;t0++) {
110   #pragma omp barrier
      #pragma omp for
      for (int ilpp=t0-N;ilpp<=t0-1;ilpp++) {
        y[t0-ilpp-1] = y[t0-ilpp-1] + A[t0-N-1][t0-ilpp-1] * tmp[t0-N-1];
        tmp[ilpp] = tmp[ilpp] + A[ilpp][t0-ilpp-1] * x[t0-ilpp-1];
115   }
      #pragma omp master
      for (int ilpp=t0;ilpp<=t0;ilpp++) {
        tmp[t0] = 0;
      }
120 }
  }
  for (int t0=max(M,N+1);t0<=N+M-1;t0++) {
    #pragma omp barrier
    #pragma omp for
125 for (int ilpp=t0-N;ilpp<=M-1;ilpp++) {
      y[t0-ilpp-1] = y[t0-ilpp-1] + A[t0-N-1][t0-ilpp-1] * tmp[t0-N-1];
      tmp[ilpp] = tmp[ilpp] + A[ilpp][t0-ilpp-1] * x[t0-ilpp-1];
    }
    #pragma omp barrier
```

```
      #pragma omp for
      for (int ilpp=M;ilpp<=t0-1;ilpp++) {
        y[t0-ilpp-1] = y[t0-ilpp-1] + A[t0-N-1][t0-ilpp-1] * tmp[t0-N-1];
      }
    }
    if ((M >= 1) && (N >= 1)) {
      #pragma omp barrier
      #pragma omp for
      for (int ilpp=M;ilpp<=N+M-1;ilpp++) {
        y[-ilpp+N+M-1] = y[-ilpp+N+M-1] + A[M-1][-ilpp+N+M-1] * tmp[M-1];
      }
    }
  }
}
```

Листинг B.10 atax (параллельный вариант ilpy без директив OpenMP, синхронный параллелизм) на языке C

```
if (N >= 1) {
 for (ilpp=0;ilpp<=N-1;ilpp++) {
  y[-ilpp+N-1] = 0;
 }
}
if ((M >= 1) && (N >= 1)) {
 for (ilpp=0;ilpp<=0;ilpp++) {
  tmp = 0;
 }
}
if (N <= 0) {
 for (t0=1;t0<=3*M-2;t0++) {
  for (ilpp=(t0-1)/3;ilpp<=(t0-1)/3;ilpp++) {
   if ((t0+2)%3 == 0) {
    tmp = 0;
   }
  }
 }
}
if ((M == 1) && (N >= 1)) {
 for (t1=0;t1<=N-1;t1++) {
  for (ilpp=0;ilpp<=0;ilpp++) {
   tmp = tmp + A[0][t1] * x[t1];
  }
 }
}
if ((M >= 2) && (N >= 1)) {
 for (t1=0;t1<=N-1;t1++) {
  for (ilpp=0;ilpp<=0;ilpp++) {
   tmp = tmp + A[0][t1] * x[t1];
  }
 }
}
if (N >= 1) {
 for (t0=3;t0<=3*M-2;t0++) {
  for (ilpp=(t0-2)/3;ilpp<=(t0-2)/3;ilpp++) {
   if ((t0+1)%3 == 0) {
    tmp = tmp + A[(t0-2)/3][0] * x[0];
   }
  }
  for (ilpp=(t0-1)/3;ilpp<=(t0-1)/3;ilpp++) {
   if ((t0+2)%3 == 0) {
```

```
    tmp = 0;
   }
  }
  for (ilpp=ceild(t0,3);ilpp<=floord(t0+3*N-3,3);ilpp++) {
   if (t0%3 == 0) {
    y[(t0-3*ilpp+3*N-3)/3] = y[(t0-3*ilpp+3*N-3)/3] + A[(t0-3)/3][(t0-3*ilpp+3*N-3)/3] * tmp;
   }
  }
  for (t1=1;t1<=N-1;t1++) {
   for (ilpp=(t0-2)/3;ilpp<=(t0-2)/3;ilpp++) {
    if ((t0+1)%3 == 0) {
     tmp = tmp + A[(t0-2)/3][t1] * x[t1];
    }
   }
  }
 }
}
if ((M >= 2) && (N >= 1)) {
 for (t1=0;t1<=N-1;t1++) {
  for (ilpp=M-1;ilpp<=M-1;ilpp++) {
   tmp = tmp + A[M-1][t1] * x[t1];
  }
 }
}
if ((M >= 1) && (N >= 1)) {
 for (ilpp=M;ilpp<=N+M-1;ilpp++) {
  y[-ilpp+N+M-1] = y[-ilpp+N+M-1] + A[M-1][-ilpp+N+M-1] * tmp;
 }
}
```

Листинг B.11 atax_ilp_sync (параллельный вариант ilpy с директивами OpenMP, синхронный параллелизм) на языке C

```
void atax_ilp_sync(int M, int N, double** A, double* x, double* y, double* tmp) {
  #pragma omp parallel
  {
    if (N >= 1) {
      #pragma omp barrier
      #pragma omp for
      for (int ilpp=0;ilpp<=N-1;ilpp++) {
        y[-ilpp+N-1] = 0;
      }
    }
    if ((M >= 1) && (N >= 1)) {
      #pragma omp master
      for (int ilpp=0;ilpp<=0;ilpp++) {
        tmp[0] = 0;
      }
    }
    if (N <= 0) {
      for (int t0=1;t0<=3*M-2;t0++) {
        if ((t0+2)%3 == 0) {
          #pragma omp master
          for (int ilpp=(t0-1)/3;ilpp<=(t0-1)/3;ilpp++) {
            tmp[0] = 0;
          }
        }
      }
    }
    if ((M == 1) && (N >= 1)) {
```

```
    for (int t1=0;t1<=N-1;t1++) {
      #pragma omp master
      for (int ilpp=0;ilpp<=0;ilpp++) {
        tmp[0] = tmp[0] + A[0][t1] * x[t1];
      }
    }
  }
  if ((M >= 2) && (N >= 1)) {
    for (int t1=0;t1<=N-1;t1++) {
      #pragma omp master
      for (int ilpp=0;ilpp<=0;ilpp++) {
        tmp[0] = tmp[0] + A[0][t1] * x[t1];
      }
    }
  }
  if (N >= 1) {
    for (int t0=3;t0<=3*M-2;t0++) {
      if ((t0+1)%3 == 0) {
        #pragma omp master
        for (int ilpp=(t0-2)/3;ilpp<=(t0-2)/3;ilpp++) {
          tmp[0] = tmp[0] + A[(t0-2)/3][0] * x[0];
        }
      } else if ((t0+2)%3 == 0) {
        #pragma omp master
        for (int ilpp=(t0-1)/3;ilpp<=(t0-1)/3;ilpp++) {
          tmp[0] = 0;
        }
      } else if (t0%3 == 0) {
        #pragma omp barrier
        #pragma omp for
        for (int ilpp=ceild(t0,3);ilpp<=floord(t0+3*N-3,3);ilpp++) {
          y[(t0-3*ilpp+3*N-3)/3] = y[(t0-3*ilpp+3*N-3)/3] + A[(t0-3)/3][(t0-3*ilpp+3*N-3)/3] *
  ↪ tmp[0];
        }
      }
      if ((t0+1)%3 == 0) {
        for (int t1=1;t1<=N-1;t1++) {
          #pragma omp master
          for (int ilpp=(t0-2)/3;ilpp<=(t0-2)/3;ilpp++) {
            tmp[0] = tmp[0] + A[(t0-2)/3][t1] * x[t1];
          }
        }
      }
    }
  }
  if ((M >= 2) && (N >= 1)) {
    for (int t1=0;t1<=N-1;t1++) {
      #pragma omp master
      for (int ilpp=M-1;ilpp<=M-1;ilpp++) {
        tmp[0] = tmp[0] + A[M-1][t1] * x[t1];
      }
    }
  }
  if ((M >= 1) && (N >= 1)) {
    #pragma omp barrier
    #pragma omp for
    for (int ilpp=M;ilpp<=N+M-1;ilpp++) {
      y[-ilpp+N+M-1] = y[-ilpp+N+M-1] + A[M-1][-ilpp+N+M-1] * tmp[0];
    }
  }
}
```

```
}
```

Листинг B.12 atax_p (параллельный вариант ilpy без директив OpenMP, расписание на основе топологической сортировки, синхронный параллелизм) на языке C

```
if ((M >= 0) && (N >= 0)) {
 for (ilpp=0;ilpp<=N+M-1;ilpp++) {
  if (ilpp <= N-1) {
   y[-ilpp+N-1] = 0;
  }
  if (ilpp <= M-1) {
   tmp[ilpp] = 0;
  }
 }
}
if (M <= -1) {
 for (ilpp=0;ilpp<=N-1;ilpp++) {
  y[-ilpp+N-1] = 0;
 }
}
if (N <= -1) {
 for (ilpp=0;ilpp<=M-1;ilpp++) {
  tmp[ilpp] = 0;
 }
}
if (M >= 1) {
 for (t2=0;t2<=N-1;t2++) {
  for (ilpp=0;ilpp<=M-1;ilpp++) {
   tmp[ilpp] = tmp[ilpp] + A[ilpp][t2] * x[t2];
  }
 }
}
if (N >= 1) {
 for (t2=0;t2<=M-1;t2++) {
  for (ilpp=t2+1;ilpp<=t2+N;ilpp++) {
   y[t2-ilpp+N] = y[t2-ilpp+N] + A[t2][t2-ilpp+N] * tmp[t2];
  }
 }
}
```

Листинг B.13 atax_p_ilp_sync_mdp (параллельный вариант ilpy с директивами OpenMP, расписание на основе топологической сортировки, синхронный параллелизм) на языке C

```
void atax_p_ilp_sync_mdp(int M, int N, double** A, double* x, double* y, double* tmp) {
  #pragma omp parallel
  {
    if ((M >= 0) && (N >= 0)) {
      #pragma omp barrier
      #pragma omp for
      for (int ilpp=0;ilpp<=N+M-1;ilpp++) {
        if (ilpp <= N-1) {
          y[-ilpp+N-1] = 0;
        }
        if (ilpp <= M-1) {
          tmp[ilpp] = 0;
```

```
        }
      }
    }
    if (M <= -1) {
      #pragma omp barrier
      #pragma omp for
      for (int ilpp=0;ilpp<=N-1;ilpp++) {
        y[-ilpp+N-1] = 0;
      }
    }
    if (N <= -1) {
      #pragma omp barrier
      #pragma omp for
      for (int ilpp=0;ilpp<=M-1;ilpp++) {
        tmp[ilpp] = 0;
      }
    }
    if (M >= 1) {
      #pragma omp barrier
      #pragma omp for
      for (int ilpp=0;ilpp<=M-1;ilpp++) {
        for (int t2=0;t2<=N-1;t2++) {
          tmp[ilpp] = tmp[ilpp] + A[ilpp][t2] * x[t2];
        }
      }
    }
    if (N >= 1) {
      for (int t2=0;t2<=M-1;t2++) {
        #pragma omp barrier
        #pragma omp for
        for (int ilpp=t2+1;ilpp<=t2+N;ilpp++) {
          y[t2-ilpp+N] = y[t2-ilpp+N] + A[t2][t2-ilpp+N] * tmp[t2];
        }
      }
    }
  }

}
```

Листинг B.14 atax_p (параллельный вариант ilpy без конструкций MPI, синхронный параллелизм) на языке C

```
if ((M <= 0) && (N >= 1)) {
 for (ilpp=0;ilpp<=N-1;ilpp++) {
  y[ilpp] = 0;
 }
}
if ((M >= 1) && (N == 0)) {
 for (ilpp=0;ilpp<=0;ilpp++) {
  tmp[0] = 0;
 }
}
if ((M >= 1) && (N >= 1)) {
 for (ilpp=0;ilpp<=0;ilpp++) {
  tmp[0] = 0;
 }
}
if (N <= 0) {
 for (t0=max(0,N+1);t0<=M-1;t0++) {
  for (ilpp=t0;ilpp<=t0;ilpp++) {
```

```
   tmp[t0] = 0;
  }
 }
}
for (t0=1;t0<=min(M-1,N-1);t0++) {
 for (ilpp=0;ilpp<=t0-1;ilpp++) {
  tmp[t0-ilpp-1] = tmp[t0-ilpp-1] + A[t0-ilpp-1][ilpp] * x[ilpp];
 }
 for (ilpp=t0;ilpp<=t0;ilpp++) {
  tmp[t0] = 0;
 }
}
if (M >= 1) {
 for (t0=M;t0<=N-1;t0++) {
  for (ilpp=t0-M;ilpp<=t0-1;ilpp++) {
   tmp[t0-ilpp-1] = tmp[t0-ilpp-1] + A[t0-ilpp-1][ilpp] * x[ilpp];
  }
 }
}
if ((N >= 1) && (N <= M-1)) {
 for (ilpp=0;ilpp<=N-1;ilpp++) {
  if (2*ilpp == N-1) {
   if ((N+1)%2 == 0) {
    y[(N-1)/2] = 0;
    tmp[(N-1)/2] = tmp[(N-1)/2] + A[(N-1)/2][(N-1)/2] * x[(N-1)/2];
   }
  }
  if (ilpp <= floord(N-2,2)) {
   y[ilpp] = 0;
  }
  if (ilpp <= floord(N-2,2)) {
   tmp[-ilpp+N-1] = tmp[-ilpp+N-1] + A[-ilpp+N-1][ilpp] * x[ilpp];
  }
  if (ilpp >= ceild(N,2)) {
   tmp[-ilpp+N-1] = tmp[-ilpp+N-1] + A[-ilpp+N-1][ilpp] * x[ilpp];
  }
  if (ilpp >= ceild(N,2)) {
   y[ilpp] = 0;
  }
 }
 for (ilpp=N;ilpp<=N;ilpp++) {
  tmp[N] = 0;
 }
}
if ((M >= 1) && (N >= M)) {
 for (ilpp=0;ilpp<=N-M-1;ilpp++) {
  y[ilpp] = 0;
 }
 for (ilpp=N-M;ilpp<=N-1;ilpp++) {
  if (2*ilpp == N-1) {
   if ((N+1)%2 == 0) {
    y[(N-1)/2] = 0;
    tmp[(N-1)/2] = tmp[(N-1)/2] + A[(N-1)/2][(N-1)/2] * x[(N-1)/2];
   }
  }
  if (ilpp <= floord(N-2,2)) {
   y[ilpp] = 0;
  }
  if (ilpp >= ceild(N,2)) {
   tmp[-ilpp+N-1] = tmp[-ilpp+N-1] + A[-ilpp+N-1][ilpp] * x[ilpp];
  }
```

```
  if (ilpp <= floord(N-2,2)) {
   tmp[-ilpp+N-1] = tmp[-ilpp+N-1] + A[-ilpp+N-1][ilpp] * x[ilpp];
  }
  if (ilpp >= ceild(N,2)) {
   y[ilpp] = 0;
  }
 }
}
if (N >= 1) {
 for (t0=N+1;t0<=M-1;t0++) {
  for (ilpp=0;ilpp<=N-1;ilpp++) {
   y[ilpp] = y[ilpp] + A[t0-N-1][ilpp] * tmp[t0-N-1];
   tmp[t0-ilpp-1] = tmp[t0-ilpp-1] + A[t0-ilpp-1][ilpp] * x[ilpp];
  }
  for (ilpp=t0;ilpp<=t0;ilpp++) {
   tmp[t0] = 0;
  }
 }
}
for (t0=max(M,N+1);t0<=N+M-1;t0++) {
 for (ilpp=0;ilpp<=t0-M-1;ilpp++) {
  y[ilpp] = y[ilpp] + A[t0-N-1][ilpp] * tmp[t0-N-1];
 }
 for (ilpp=t0-M;ilpp<=N-1;ilpp++) {
  y[ilpp] = y[ilpp] + A[t0-N-1][ilpp] * tmp[t0-N-1];
  tmp[t0-ilpp-1] = tmp[t0-ilpp-1] + A[t0-ilpp-1][ilpp] * x[ilpp];
 }
}
if ((M >= 1) && (N >= 1)) {
 for (ilpp=0;ilpp<=N-1;ilpp++) {
  y[ilpp] = y[ilpp] + A[M-1][ilpp] * tmp[M-1];
 }
}
```

Листинг B.15 atax_p_ilp_arrays (параллельный вариант ilpy с конструкциями MPI, синхронный параллелизм) на языке C

```
void atax_p_ilp_arrays(int M, int N, double** A, double* x, double* y, double* tmp) {
  if ((M <= 0) && (N >= 1)) {
    for(int ilpp=max(lR,0);ilpp<=min(uR,N-1);ilpp++) {
      y[ilpp] = 0;
    }
    MPI_Barrier_time(MPI_COMM_WORLD);
  }
  if ((M >= 1) && (N == 0)) {
    for(int ilpp=max(lR,0);ilpp<=min(uR,0);ilpp++) {
      tmp[0] = 0;
    }
    MPI_Barrier_time(MPI_COMM_WORLD);
  }
  if ((M >= 1) && (N >= 1)) {
    for(int ilpp=max(lR,0);ilpp<=min(uR,0);ilpp++) {
      tmp[0] = 0;
    }
    MPI_Barrier_time(MPI_COMM_WORLD);
  }
  if (N <= 0) {
    for(int t0=max(0,N+1);t0<=M-1;t0++) {
      for(int ilpp=max(lR,t0);ilpp<=min(uR,t0);ilpp++) {
        tmp[t0] = 0;
```

```
    }
    MPI_Barrier_time(MPI_COMM_WORLD);
    // tmp[t0] in tmp[t0] = 0
    line_Q_write_vp_rev(0, t0, t0, t0, tmp, eta_tmp, T_tmp);
    //
  }
}
for(int t0=1;t0<=min(M-1,N-1);t0++) {
  // tmp[t0-ilpp-1] in tmp[t0-ilpp-1] = tmp[t0-ilpp-1] + A[t0-ilpp-1][ilpp] * x[ilpp]
  line_Q_read_vp_rev(-1, t0 - 1, 0, t0 - 1, tmp, eta_tmp, T_tmp);
  //
  for(int ilpp=max(lR,0);ilpp<=min(uR,t0-1);ilpp++) {
    tmp[t0-ilpp-1] = tmp[t0-ilpp-1] + A[t0-ilpp-1][ilpp] * x[ilpp];
  }
  MPI_Barrier_time(MPI_COMM_WORLD);
  // tmp[t0-ilpp-1] in tmp[t0-ilpp-1] = tmp[t0-ilpp-1] + A[t0-ilpp-1][ilpp] * x[ilpp]
  line_Q_write_vp_rev(-1, t0 - 1, 0, t0 - 1, tmp, eta_tmp, T_tmp);
  //
  for(int ilpp=max(lR,t0);ilpp<=min(uR,t0);ilpp++) {
    tmp[t0] = 0;
  }
  MPI_Barrier_time(MPI_COMM_WORLD);
  // tmp[t0] in tmp[t0] = 0
  line_Q_write_vp_rev(0, t0, t0, t0, tmp, eta_tmp, T_tmp);
  //
}
if (M >= 1) {
  for(int t0=M;t0<=N-1;t0++) {
    // tmp[t0-ilpp-1] in tmp[t0-ilpp-1] = tmp[t0-ilpp-1] + A[t0-ilpp-1][ilpp] * x[ilpp]
    line_Q_read_vp_rev(-1, t0 - 1, t0 - M, t0 - 1, tmp, eta_tmp, T_tmp);
    //
    for(int ilpp=max(lR,t0-M);ilpp<=min(uR,t0-1);ilpp++) {
      tmp[t0-ilpp-1] = tmp[t0-ilpp-1] + A[t0-ilpp-1][ilpp] * x[ilpp];
    }
    MPI_Barrier_time(MPI_COMM_WORLD);
    // tmp[t0-ilpp-1] in tmp[t0-ilpp-1] = tmp[t0-ilpp-1] + A[t0-ilpp-1][ilpp] * x[ilpp]
    line_Q_write_vp_rev(-1, t0 - 1, t0 - M, t0 - 1, tmp, eta_tmp, T_tmp);
    //
  }
}
if ((N >= 1) && (N <= M-1)) {
  // tmp[-ilpp+N-1] in tmp[-ilpp+N-1] = tmp[-ilpp+N-1] + A[-ilpp+N-1][ilpp] * x[ilpp];
  // tmp[(N-1)/2] in tmp[(N-1)/2] = tmp[(N-1)/2] + A[(N-1)/2][(N-1)/2] * x[(N-1)/2];
  line_Q_read_vp_rev(-1, N - 1, 0, N - 1, tmp, eta_tmp, T_tmp);
  //
  for(int ilpp=max(lR,0);ilpp<=min(uR,N-1);ilpp++) {
    if (2*ilpp == N-1) {
      if ((N+1)%2 == 0) {
        y[(N-1)/2] = 0;
        tmp[(N-1)/2] = tmp[(N-1)/2] + A[(N-1)/2][(N-1)/2] * x[(N-1)/2];
      }
    }
    if (ilpp <= floord(N-2,2)) {
      y[ilpp] = 0;
    }
    if (ilpp <= floord(N-2,2)) {
      tmp[-ilpp+N-1] = tmp[-ilpp+N-1] + A[-ilpp+N-1][ilpp] * x[ilpp];
    }
    if (ilpp >= ceild(N,2)) {
      tmp[-ilpp+N-1] = tmp[-ilpp+N-1] + A[-ilpp+N-1][ilpp] * x[ilpp];
    }
```

```
      if (ilpp >= ceild(N,2)) {
        y[ilpp] = 0;
      }
    }
    MPI_Barrier_time(MPI_COMM_WORLD);
    // tmp[-ilpp+N-1] in tmp[-ilpp+N-1] = tmp[-ilpp+N-1] + A[-ilpp+N-1][ilpp] * x[ilpp];
    // tmp[(N-1)/2] in tmp[(N-1)/2] = tmp[(N-1)/2] + A[(N-1)/2][(N-1)/2] * x[(N-1)/2];
    line_Q_write_vp_rev(-1, N - 1, 0, N - 1, tmp, eta_tmp, T_tmp);
    //
    for(int ilpp=max(lR,N);ilpp<=min(uR,N);ilpp++) {
      tmp[N] = 0;
    }
    MPI_Barrier_time(MPI_COMM_WORLD);
    // tmp[N] in tmp[N] = 0
    line_Q_write_vp_rev(0, N, N, N, tmp, eta_tmp, T_tmp);
    //
  }
  if ((M >= 1) && (N >= M)) {
    for(int ilpp=max(lR,0);ilpp<=min(uR,N-M-1);ilpp++) {
      y[ilpp] = 0;
    }
    MPI_Barrier_time(MPI_COMM_WORLD);
    // tmp[-ilpp+N-1] in tmp[-ilpp+N-1] = tmp[-ilpp+N-1] + A[-ilpp+N-1][ilpp] * x[ilpp];
    // tmp[(N-1)/2] in tmp[(N-1)/2] = tmp[(N-1)/2] + A[(N-1)/2][(N-1)/2] * x[(N-1)/2];
    line_Q_read_vp_rev(-1, N - 1, N - M, N - 1, tmp, eta_tmp, T_tmp);
    //
    for(int ilpp=max(lR,N-M);ilpp<=min(uR,N-1);ilpp++) {
      if (2*ilpp == N-1) {
        if ((N+1)%2 == 0) {
          y[(N-1)/2] = 0;
          tmp[(N-1)/2] = tmp[(N-1)/2] + A[(N-1)/2][(N-1)/2] * x[(N-1)/2];
        }
      }
      if (ilpp <= floord(N-2,2)) {
        y[ilpp] = 0;
      }
      if (ilpp >= ceild(N,2)) {
        tmp[-ilpp+N-1] = tmp[-ilpp+N-1] + A[-ilpp+N-1][ilpp] * x[ilpp];
      }
      if (ilpp <= floord(N-2,2)) {
        tmp[-ilpp+N-1] = tmp[-ilpp+N-1] + A[-ilpp+N-1][ilpp] * x[ilpp];
      }
      if (ilpp >= ceild(N,2)) {
        y[ilpp] = 0;
      }
    }
    MPI_Barrier_time(MPI_COMM_WORLD);
    // tmp[-ilpp+N-1] in tmp[-ilpp+N-1] = tmp[-ilpp+N-1] + A[-ilpp+N-1][ilpp] * x[ilpp];
    // tmp[(N-1)/2] in tmp[(N-1)/2] = tmp[(N-1)/2] + A[(N-1)/2][(N-1)/2] * x[(N-1)/2];
    line_Q_write_vp_rev(-1, N - 1, N - M, N - 1, tmp, eta_tmp, T_tmp);
    //
  }
  if (N >= 1) {
    for(int t0=N+1;t0<=M-1;t0++) {
      // tmp[t0-ilpp-1] in tmp[t0-ilpp-1] = tmp[t0-ilpp-1] + A[t0-ilpp-1][ilpp] * x[ilpp]
      line_Q_read_vp_rev(-1, t0 - 1, 0, N - 1, tmp, eta_tmp, T_tmp);
      //
      // tmp[t0-N-1] in y[ilpp] = y[ilpp] + A[t0-N-1][ilpp] * tmp[t0-N-1]
      line_Q_read_vp_rev(0, t0 - N - 1, 0, N - 1, tmp, eta_tmp, T_tmp);
      //
      for(int ilpp=max(lR,0);ilpp<=min(uR,N-1);ilpp++) {
```

```
        y[ilpp] = y[ilpp] + A[t0-N-1][ilpp] * tmp[t0-N-1];
        tmp[t0-ilpp-1] = tmp[t0-ilpp-1] + A[t0-ilpp-1][ilpp] * x[ilpp];
      }
      MPI_Barrier_time(MPI_COMM_WORLD);
      // tmp[t0-ilpp-1] in tmp[t0-ilpp-1] = tmp[t0-ilpp-1] + A[t0-ilpp-1][ilpp] * x[ilpp]
      line_Q_write_vp_rev(-1, t0 - 1, 0, N - 1, tmp, eta_tmp, T_tmp);
      //
      for(int ilpp=max(lR,t0);ilpp<=min(uR,t0);ilpp++) {
        tmp[t0] = 0;
      }
      MPI_Barrier_time(MPI_COMM_WORLD);
      // tmp[t0] in tmp[t0] = 0
      line_Q_write_vp_rev(0, t0, t0, t0, tmp, eta_tmp, T_tmp);
      //
    }
  }
  for(int t0=max(M,N+1);t0<=N+M-1;t0++) {
    // tmp[t0-N-1] in y[ilpp] = y[ilpp] + A[t0-N-1][ilpp] * tmp[t0-N-1]
    line_Q_read_vp_rev(0, t0 - N - 1, 0, t0 - M - 1, tmp, eta_tmp, T_tmp);
    //
    for(int ilpp=max(lR,0);ilpp<=min(uR,t0-M-1);ilpp++) {
      y[ilpp] = y[ilpp] + A[t0-N-1][ilpp] * tmp[t0-N-1];
    }
    MPI_Barrier_time(MPI_COMM_WORLD);
    // tmp[t0-ilpp-1] in tmp[t0-ilpp-1] = tmp[t0-ilpp-1] + A[t0-ilpp-1][ilpp] * x[ilpp]
    line_Q_read_vp_rev(-1, t0 - 1, t0 - M, N - 1, tmp, eta_tmp, T_tmp);
    //
    // tmp[t0-N-1] in y[ilpp] = y[ilpp] + A[t0-N-1][ilpp] * tmp[t0-N-1];
    line_Q_read_vp_rev(0, t0 - N - 1, t0 - M, N - 1, tmp, eta_tmp, T_tmp);
    //
    for(int ilpp=max(lR,t0-M);ilpp<=min(uR,N-1);ilpp++) {
      y[ilpp] = y[ilpp] + A[t0-N-1][ilpp] * tmp[t0-N-1];
      tmp[t0-ilpp-1] = tmp[t0-ilpp-1] + A[t0-ilpp-1][ilpp] * x[ilpp];
    }
    MPI_Barrier_time(MPI_COMM_WORLD);
    // tmp[t0-ilpp-1] in tmp[t0-ilpp-1] = tmp[t0-ilpp-1] + A[t0-ilpp-1][ilpp] * x[ilpp]
    line_Q_write_vp_rev(-1, t0 - 1, t0 - M, N - 1, tmp, eta_tmp, T_tmp);
    //
  }
  if ((M >= 1) && (N >= 1)) {
    // tmp[M-1] in y[ilpp] = y[ilpp] + A[M-1][ilpp] * tmp[M-1]
    line_Q_read_vp_rev(0, M - 1, 0, N - 1, tmp, eta_tmp, T_tmp);
    //
    for(int ilpp=max(lR,0);ilpp<=min(uR,N-1);ilpp++) {
      y[ilpp] = y[ilpp] + A[M-1][ilpp] * tmp[M-1];
    }
    MPI_Barrier_time(MPI_COMM_WORLD);
  }
  collect_matrix_cols_double(&y, 1, N, y_chunk_size, false);
}
```

Листинг B.16 atax_p (параллельный вариант ilpy без конструкций MPI, расписание на основе топологической сортировки, синхронный параллелизм) на языке C

```
if ((M >= 0) && (N >= 0)) {
 for (ilpp=0;ilpp<=N+M-1;ilpp++) {
  if (ilpp <= N-1) {
   y[-ilpp+N-1] = 0;
```

```
  }
  if (ilpp <= M-1) {
   tmp[-ilpp+M-1] = 0;
  }
 }
}
if (M <= -1) {
 for (ilpp=0;ilpp<=N-1;ilpp++) {
  y[-ilpp+N-1] = 0;
 }
}
if (N <= -1) {
 for (ilpp=0;ilpp<=M-1;ilpp++) {
  tmp[-ilpp+M-1] = 0;
 }
}
if (M >= 1) {
 for (t2=0;t2<=N-1;t2++) {
  for (ilpp=0;ilpp<=M-1;ilpp++) {
   tmp[-ilpp+M-1] = tmp[-ilpp+M-1] + A[-ilpp+M-1][t2] * x[t2];
  }
 }
}
if (N >= 1) {
 for (t2=0;t2<=M-1;t2++) {
  for (ilpp=0;ilpp<=N-1;ilpp++) {
   y[-ilpp+N-1] = y[-ilpp+N-1] + A[t2][-ilpp+N-1] * tmp[t2];
  }
 }
}
```

Листинг B.17 atax_p_ilp_arrays_mdp (параллельный вариант ilpy с конструкциями MPI, расписание на основе топологической сортировки, синхронный параллелизм) на языке C

```
void atax_p_ilp_arrays_mdp(int M, int N, double** A, double* x, double* y, double* tmp, bool
    ↪ use_bcast, bool dist_input) {
  if ((M >= 0) && (N >= 0)) {
    for (int ilpp=max(lR,0);ilpp<=min(uR,N+M-1);ilpp++) {
      if (ilpp <= N-1) {
       y[-ilpp+N-1] = 0;
      }
      if (ilpp <= M-1) {
       tmp[-ilpp+M-1] = 0;
      }
    }
    MPI_Barrier_time(MPI_COMM_WORLD);
  }
  if (M <= -1) {
    for (int ilpp=max(lR,0);ilpp<=min(uR,N-1);ilpp++) {
      y[-ilpp+N-1] = 0;
    }
    MPI_Barrier_time(MPI_COMM_WORLD);
  }
  if (N <= -1) {
    for (int ilpp=max(lR,0);ilpp<=min(uR,M-1);ilpp++) {
      tmp[-ilpp+M-1] = 0;
    }
    MPI_Barrier_time(MPI_COMM_WORLD);
```

```
  }
  if (M >= 1) {
    if (dist_input) {
      // x[t2] in tmp[-ilpp+M-1] = tmp[-ilpp+M-1] + A[-ilpp+M-1][t2] * x[t2]
      if (use_bcast) {
        IF_TIME(mpi_spent_time -= rtclock());
        MPI_Bcast(x, N, MPI_DOUBLE, blockdist_proc_rank(M-1), MPI_COMM_WORLD);
        IF_TIME(mpi_spent_time += rtclock());
      }
      else {
        line_Q_read_send(0, clamp(lr, 0, M-1), N-1, clamp(ur, 0, M-1), x, eta_x_mdp, T_x);
        line_Q_read_receive(0, clamp(lR, 0, M-1), N-1, clamp(uR, 0, M-1), x, eta_x_mdp, T_x);
      }
      //
      // A[-ilpp+M-1][t2] in tmp[-ilpp+M-1] = tmp[-ilpp+M-1] + A[-ilpp+M-1][t2] * x[t2]
      for (int ilpp=0;ilpp<=M-1;ilpp++) {
        if (use_bcast) {
          IF_TIME(mpi_spent_time -= rtclock());
          for (int r = 0; r < Q; r++) {
            int actual_chunk_size = blockdist_array_distribute_r(N, r);
            int chunk_start = blockdist_chunk_start_inv_r(N, A_chunk_size, actual_chunk_size, r);
            MPI_Bcast(A[-ilpp+M-1] + chunk_start, actual_chunk_size, MPI_DOUBLE, r, MPI_COMM_WORLD);
          }
          IF_TIME(mpi_spent_time += rtclock());
        }
        else {
          line_Q_read_send(blockdist_chunk_start_inv(N, A_chunk_size, A_actual_chunk_size),
    ↪ clamp(lr, 0, M-1),
                    blockdist_chunk_end_inv(N, A_chunk_size, A_actual_chunk_size) - 1, clamp(ur, 0,
    ↪ M-1),
                    A[-ilpp+M-1], eta_A_mdp, T_A + (-ilpp+M-1));
          line_Q_read_receive(blockdist_chunk_start_inv_r(N, A_chunk_size,
    ↪ blockdist_array_distribute_r(N, r), r), clamp(lR, 0, M-1),
                     blockdist_chunk_end_inv_r(N, A_chunk_size, blockdist_array_distribute_r(N, r),
    ↪ r) - 1, clamp(uR, 0, M-1),
                     A[-ilpp+M-1], eta_A_mdp, T_A + (-ilpp+M-1));
        }
      }
      //
    }
    for (int ilpp=max(lR, 0);ilpp<=min(uR,M-1);ilpp++) {
      for (int t2=0;t2<=N-1;t2++) {
        tmp[-ilpp+M-1] = tmp[-ilpp+M-1] + A[-ilpp+M-1][t2] * x[t2];
      }
    }
    MPI_Barrier_time(MPI_COMM_WORLD);
  }
  if (N >= 1) {
    // tmp[t2] in y[-ilpp+N-1] = y[-ilpp+N-1] + A[t2][-ilpp+N-1] * tmp[t2];
    if (use_bcast) {
      IF_TIME(mpi_spent_time -= rtclock());
      for (int r = 0; r < Q; r++) {
        int actual_chunk_size = blockdist_array_distribute_r(M, r);
        int chunk_start = blockdist_chunk_start_inv_r(M, tmp_chunk_size, actual_chunk_size, r);
        MPI_Bcast(tmp + chunk_start, actual_chunk_size, MPI_DOUBLE, r, MPI_COMM_WORLD);
      }
      IF_TIME(mpi_spent_time += rtclock());
    }
    else {
      line_Q_read_send(blockdist_chunk_start_inv(M, tmp_chunk_size, tmp_actual_chunk_size),
    ↪ clamp(lr, 0, N-1),
```

```
              blockdist_chunk_end_inv(M, tmp_chunk_size, tmp_actual_chunk_size) - 1, clamp(ur, 0,
    ↪ N-1),
              tmp, eta_tmp_mdp, T_tmp);
      line_Q_read_receive(blockdist_chunk_start_inv_r(M, tmp_chunk_size,
    ↪ blockdist_array_distribute_r(M, r), r), clamp(lR, 0, N-1),
               blockdist_chunk_end_inv_r(M, tmp_chunk_size, blockdist_array_distribute_r(M, r), r)
    ↪ - 1, clamp(uR, 0, N-1),
               tmp, eta_tmp_mdp, T_tmp);
    }
    //
    int lb = max(lR, 0);
    int ub = min(uR, N-1);
    for (int t2=0;t2<=M-1;t2++) {
      // tmp[t2] in y[-ilpp+N-1] = y[-ilpp+N-1] + A[t2][-ilpp+N-1] * tmp[t2];
      //line_Q_read_send(t2, clamp(lr, 0, N-1), t2, clamp(ur, 0, N-1), tmp, eta_tmp_mdp, T_tmp,
    ↪ mpi_requests);
      //line_Q_read_receive(t2, clamp(lR, 0, N-1), t2, clamp(uR, 0, N-1), tmp, eta_tmp_mdp, T_tmp,
    ↪ mpi_statuses);
      //
      for (int ilpp=lb;ilpp<=ub;ilpp++) {
        y[-ilpp+N-1] = y[-ilpp+N-1] + A[t2][-ilpp+N-1] * tmp[t2];
      }
      MPI_Barrier_time(MPI_COMM_WORLD);
    }
  }
  collect_matrix_cols_double(&y, 1, N, y_chunk_size, true);
}
```

Листинг B.18 Размещение данных для atax_p_ilp_arrays и atax_p_ilp_arrays_mdp

```
#define T_base 1000000

#define eta_tmp(i0) 0
#define eta_tmp_mdp(i0) (-(i0)+M-1)
#define T_tmp (1 * T_base)

int tmp_chunk_size;
int tmp_actual_chunk_size;

#define eta_x(i0) (i0)
#define eta_x_mdp(i0) (M-1)
#define T_x (2 * T_base)

int x_chunk_size;
int x_actual_chunk_size;

#define eta_y(i0) (i0)
#define eta_y_mdp(i0) (-(i0)+N-1)
#define T_y (3 * T_base)

int y_chunk_size;
int y_actual_chunk_size;

#define eta_A(i1) (i1)
#define eta_A_mdp(i1) (-(i1)+N-1)
#define T_A (4 * T_base)

int A_chunk_size;
int A_actual_chunk_size;
```

B.4 Преобразования pluto

Листинг B.19 Преобразования pluto в формате cloog для atax_p_pluto

```
# Number of scattering functions
4

# T(S1)
3 8
0 1 0 0 0 0 0 -2
0 0 1 0 -1 0 0 0
0 0 0 1 0 0 0 0

# T(S2)
3 8
0 1 0 0 0 0 0 0
0 0 1 0 -1 0 0 0
0 0 0 1 0 0 0 0

# T(S3)
3 9
0 1 0 0 0 0 0 0 -1
0 0 1 0 -1 0 0 0 0
0 0 0 1 0 -1 0 0 0

# T(S4)
3 9
0 1 0 0 0 0 0 0 -3
0 0 1 0 -1 -1 0 0 0
0 0 0 1 0 -1 0 0 0
```

Листинг B.20 atax_p (параллельный вариант pluto с директивами OpenMP, синхронный параллелизм) на языке C

```
 int t1, t2, t3;
 int lb, ub, lbd, ubd, lb2, ub2;
 register int lbv, ubv;
/* Start of CLooG code */
{/* extra braces to avoid redefination of lbp*/
int lbp=0;
int ubp=M-1;
#pragma omp parallel for private(lbv,ubv,t3)
for (t2=lbp;t2<=ubp;t2++) {
 tmp[t2] = 0;;
}
}/*end of omp parallel loop */
if (N >= 1) {
 {/* extra braces to avoid redefination of lbp*/
int lbp=0;
 int ubp=M-1;
#pragma omp parallel for private(lbv,ubv,t3)
 for (t2=lbp;t2<=ubp;t2++) {
  for (t3=0;t3<=N-1;t3++) {
   tmp[t2] = tmp[t2] + A[t2][t3] * x[t3];;
  }
 }
```

```
}/*end of omp parallel loop */
}
{/* extra braces to avoid redefination of lbp*/
int lbp=0;
int ubp=N-1;
#pragma omp parallel for private(lbv,ubv,t3)
for (t2=lbp;t2<=ubp;t2++) {
 y[t2] = 0;;
}
}/*end of omp parallel loop */
if ((M >= 1) && (N >= 1)) {
 for (t2=0;t2<=N+M-2;t2++) {
  {/* extra braces to avoid redefination of lbp*/
int lbp=max(0,t2-M+1);
  int ubp=min(t2,N-1);
#pragma omp parallel for private(lbv,ubv)
  for (t3=lbp;t3<=ubp;t3++) {
   y[t3] = y[t3] + A[(t2-t3)][t3] * tmp[(t2-t3)];;
  }
}/*end of omp parallel loop */
 }
}
/* End of CLooG code */
```

Листинг B.21 atax_p_pluto (обработанный вариант pluto с директивами OpenMP, синхронный параллелизм) на языке C

```
void atax_p_pluto(int M, int N, double** A, double* x, double* y, double* tmp) {
  #pragma omp parallel
  {
    /* Start of CLooG code */
    #pragma omp for
    for (int t2=0;t2<=M-1;t2++) {
      tmp[t2] = 0;
    }
    if (N >= 1) {
      #pragma omp for
      for (int t2=0;t2<=M-1;t2++) {
        for (int t3=0;t3<=N-1;t3++) {
          tmp[t2] = tmp[t2] + A[t2][t3] * x[t3];
        }
      }
    }
    #pragma omp for
    for (int t2=0;t2<=N-1;t2++) {
      y[t2] = 0;
    }
    if ((M >= 1) && (N >= 1)) {
      for (int t2=0;t2<=N+M-2;t2++) {
        #pragma omp for
        for (int t3=max(0,t2-M+1);t3<=min(t2,N-1);t3++) {
          y[t3] = y[t3] + A[(t2-t3)][t3] * tmp[(t2-t3)];;
        }
      }
    }
    /* End of CLooG code */
  }
}
```

Листинг B.22 Преобразования pluto в формате cloog для atax_pluto

```
# Number of scattering functions
4

# T(S1)
5 10
0 1 0 0 0 0 0 0 0 0
0 0 1 0 0 0 -1 0 0 0
0 0 0 1 0 0 0 0 0 -2
0 0 0 0 1 0 0 0 0 -1
0 0 0 0 0 1 0 0 0 0

# T(S2)
5 10
0 1 0 0 0 0 0 0 0 -1
0 0 1 0 0 0 -1 0 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 -1
0 0 0 0 0 1 0 0 0 0

# T(S3)
5 11
0 1 0 0 0 0 0 0 0 0 -1
0 0 1 0 0 0 -1 0 0 0 0
0 0 0 1 0 0 0 0 0 0 -1
0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 1 0 -1 0 0 0

# T(S4)
5 11
0 1 0 0 0 0 0 0 0 0 -1
0 0 1 0 0 0 -1 0 0 0 0
0 0 0 1 0 0 0 0 0 0 -1
0 0 0 0 1 0 0 0 0 0 -1
0 0 0 0 0 1 0 -1 0 0 0
```

Листинг B.23 atax (параллельный вариант pluto с директивами OpenMP, синхронный параллелизм) на языке C

```
 int t1, t2, t3, t4, t5;
 int lb, ub, lbd, ubd, lb2, ub2;
 register int lbv, ubv;
/* Start of CLooG code */
{/* extra braces to avoid redefination of lbp*/
int lbp=0;
int ubp=N-1;
#pragma omp parallel for private(lbv,ubv,t3,t4,t5)
for (t2=lbp;t2<=ubp;t2++) {
 y[t2] = 0;;
}
}/*end of omp parallel loop */
if (N >= 1) {
 for (t2=0;t2<=M-1;t2++) {
  tmp = 0;;
  for (t5=0;t5<=N-1;t5++) {
   tmp = tmp + A[t2][t5] * x[t5];;
  }
  {/* extra braces to avoid redefination of lbp*/
int lbp=0;
```

```
  int ubp=N-1;
#pragma omp parallel for private(lbv,ubv)
  for (t5=lbp;t5<=ubp;t5++) {
   y[t5] = y[t5] + A[t2][t5] * tmp;;
  }
}/*end of omp parallel loop */
 }
}
if (N <= 0) {
 for (t2=0;t2<=M-1;t2++) {
  tmp = 0;;
 }
}
/* End of CLooG code */
```

Листинг B.24 atax_pluto (обработанный вариант pluto с директивами OpenMP, синхронный параллелизм) на языке C

```
void atax_pluto(int M, int N, double** A, double* x, double* y, double* tmp) {
  #pragma omp parallel
  {
    /* Start of CLooG code */
    #pragma omp for
    for (int t2=0;t2<=N-1;t2++) {
     y[t2] = 0;
    }
    if (N >= 1) {
      for (int t2=0;t2<=M-1;t2++) {
        #pragma omp single
        {
          tmp[0] = 0;
          for (int t5=0;t5<=N-1;t5++) {
            tmp[0] = tmp[0] + A[t2][t5] * x[t5];
          }
        }
        #pragma omp for
        for (int t5=0;t5<=N-1;t5++) {
          y[t5] = y[t5] + A[t2][t5] * tmp[0];
        }
      }
    }
    #pragma omp single
    if (N <= 0) {
      for (int t2=0;t2<=M-1;t2++) {
        tmp[0] = 0;
      }
    }
    /* End of CLooG code */
  }
}
```

Листинг B.25 Преобразования pluto в формате cloog для atax_p_pluto_mpi

```
# Number of scattering functions
16

# T(S1)
7 13
0 1 0 0 0 0 0 0 0 0 0 0 -2
```

```
0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 -1 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0

# T(S2)
7 13
0 1 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 -1 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0

# T(S3)
7 14
0 1 0 0 0 0 0 0 0 0 0 0 -1
0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 -1 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 -1 0 0 0

# T(S4)
7 14
0 1 0 0 0 0 0 0 0 0 0 0 -3
0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 -1 -1 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 -1 0 0 0
```

Листинг B.26 atax_p_pluto_mpi (параллельный вариант pluto с конструкциями MPI, синхронный параллелизм) на языке C

```
/* Start of CLooG code */
IF_TIME(t_comp_start = rtclock());
_lb_dist=0;
_ub_dist=M-1;
polyrt_loop_dist(_lb_dist, _ub_dist, nprocs, my_rank,
    ↪ &lbd_t5, &ubd_t5);
for (t5=lbd_t5;t5<=ubd_t5;t5++) {
 tmp[t5] = 0;;
}
IF_TIME(t_comp += rtclock() - t_comp_start);
IF_TIME(t_pack_start = rtclock());
_lb_dist=0;
_ub_dist=M-1;
polyrt_loop_dist(_lb_dist, _ub_dist, nprocs, my_rank,
    ↪ &lbd_t5, &ubd_t5);
for (t5=lbd_t5;t5<=ubd_t5;t5++) {
 for (__p=0; __p<nprocs; __p++) { receiver_list[__p] =
    ↪ 0; } sigma_tmp_1_1(0,t5,N,M, my_rank, nprocs,
    ↪ receiver_list); for (__p=0; __p<nprocs; __p++) {
    ↪ if (receiver_list[__p] != 0) { send_counts_tmp[__p]
```

```
    ↪ = pack_tmp_1_1(0,t5, send_buf_tmp[__p], send_counts_tmp[__p]);
    ↪ } };
}
IF_TIME(t_pack += rtclock() - t_pack_start);
if (M >= 1) {
 IF_TIME(t_comm_start = rtclock());
    ↪ MPI_Alltoall(send_counts_tmp, 1, MPI_INT,
    ↪ recv_counts_tmp, 1, MPI_INT, MPI_COMM_WORLD); req_count=0;
    ↪ for (__p=0; __p<nprocs; __p++) { if (send_counts_tmp[__p]
    ↪ >= 1) {IF_TIME(__total_count += send_counts_tmp[__p]);
    ↪ MPI_Isend(send_buf_tmp[__p], send_counts_tmp[__p],
    ↪ MPI_DOUBLE, __p, 123, MPI_COMM_WORLD, &reqs[req_count++]);}}for
    ↪ (__p=0; __p<nprocs; __p++) { if(recv_counts_tmp[__p]
    ↪ >= 1) { MPI_Irecv(recv_buf_tmp+displs_tmp[__p], recv_counts_tmp[__p],
    ↪ MPI_DOUBLE, __p, 123, MPI_COMM_WORLD, &reqs[req_count++]);}}
    ↪ MPI_Waitall(req_count, reqs, stats); for (__p=0;
    ↪ __p<nprocs; __p++) { send_counts_tmp[__p] = 0;}for
    ↪ (__p=0; __p<nprocs; __p++) { curr_displs_tmp[__p]
    ↪ = displs_tmp[__p]; }IF_TIME(t_comm += rtclock() -
    ↪ t_comm_start);;
}
IF_TIME(t_unpack_start = rtclock());
for (t5=0;t5<=M-1;t5++) {
 proc = pi_1(0,t5,N,M, nprocs); if ((my_rank != proc)
    ↪ && (recv_counts_tmp[proc] > 0)) { if
    ↪ (is_receiver_tmp_1_1(0,t5,N,M,my_rank,nprocs) !=
    ↪ 0) { curr_displs_tmp[proc] = unpack_tmp_1_1(0,t5,recv_buf_tmp,curr_displs_tmp[proc]);
    ↪ }};
}
IF_TIME(t_unpack += rtclock() - t_unpack_start);
if (N >= 1) {
 IF_TIME(t_comp_start = rtclock());
_lb_dist=0;
 _ub_dist=M-1;
 polyrt_loop_dist(_lb_dist, _ub_dist, nprocs, my_rank,
    ↪ &lbd_t5, &ubd_t5);
 for (t5=lbd_t5;t5<=ubd_t5;t5++) {
  lbv=0;
  ubv=N-1;
#pragma ivdep
#pragma vector always
  for (t7=lbv;t7<=ubv;t7++) {
   tmp[t5] = tmp[t5] + A[t5][t7] * x[t7];;
  }
 }
IF_TIME(t_comp += rtclock() - t_comp_start);
}
if (N >= 1) {
 IF_TIME(t_pack_start = rtclock());
_lb_dist=0;
 _ub_dist=M-1;
 polyrt_loop_dist(_lb_dist, _ub_dist, nprocs, my_rank,
    ↪ &lbd_t5, &ubd_t5);
 for (t5=lbd_t5;t5<=ubd_t5;t5++) {
  for (__p=0; __p<nprocs; __p++) { receiver_list[__p]
    ↪ = 0; } sigma_tmp_1_2(1,t5,N,M, my_rank, nprocs,
    ↪ receiver_list); for (__p=0; __p<nprocs; __p++) {
    ↪ if (receiver_list[__p] != 0) { send_counts_tmp[__p]
    ↪ = pack_tmp_1_2(1,t5, send_buf_tmp[__p], send_counts_tmp[__p]);
    ↪ } };
 }
```

```
IF_TIME(t_pack += rtclock() - t_pack_start);
}
if ((M >= 1) && (N >= 1)) {
 IF_TIME(t_comm_start = rtclock());
    ↪ MPI_Alltoall(send_counts_tmp, 1, MPI_INT,
    ↪ recv_counts_tmp, 1, MPI_INT, MPI_COMM_WORLD); req_count=0;
    ↪ for (__p=0; __p<nprocs; __p++) { if (send_counts_tmp[__p]
    ↪ >= 1) {IF_TIME(__total_count += send_counts_tmp[__p]);
    ↪ MPI_Isend(send_buf_tmp[__p], send_counts_tmp[__p],
    ↪ MPI_DOUBLE, __p, 123, MPI_COMM_WORLD, &reqs[req_count++]);}}for
    ↪ (__p=0; __p<nprocs; __p++) { if(recv_counts_tmp[__p]
    ↪ >= 1) { MPI_Irecv(recv_buf_tmp+displs_tmp[__p], recv_counts_tmp[__p],
    ↪ MPI_DOUBLE, __p, 123, MPI_COMM_WORLD, &reqs[req_count++]);}}
    ↪ MPI_Waitall(req_count, reqs, stats); for (__p=0;
    ↪ __p<nprocs; __p++) { send_counts_tmp[__p] = 0;}for
    ↪ (__p=0; __p<nprocs; __p++) { curr_displs_tmp[__p]
    ↪ = displs_tmp[__p]; }IF_TIME(t_comm += rtclock() -
    ↪ t_comm_start);;
}
if (N >= 1) {
 IF_TIME(t_unpack_start = rtclock());
for (t5=0;t5<=M-1;t5++) {
  proc = pi_2(1,t5,N,M, nprocs); if ((my_rank != proc)
    ↪ && (recv_counts_tmp[proc] > 0)) { if
    ↪ (is_receiver_tmp_1_2(1,t5,N,M,my_rank,nprocs) !=
    ↪ 0) { curr_displs_tmp[proc] = unpack_tmp_1_2(1,t5,recv_buf_tmp,curr_displs_tmp[proc]);
    ↪ }};
 }
IF_TIME(t_unpack += rtclock() - t_unpack_start);
}
IF_TIME(t_comp_start = rtclock());
_lb_dist=0;
_ub_dist=N-1;
polyrt_loop_dist(_lb_dist, _ub_dist, nprocs, my_rank,
    ↪ &lbd_t5, &ubd_t5);
for (t5=lbd_t5;t5<=ubd_t5;t5++) {
 y[t5] = 0;;
}
IF_TIME(t_comp += rtclock() - t_comp_start);
IF_TIME(t_pack_start = rtclock());
_lb_dist=0;
_ub_dist=N-1;
polyrt_loop_dist(_lb_dist, _ub_dist, nprocs, my_rank,
    ↪ &lbd_t5, &ubd_t5);
for (t5=lbd_t5;t5<=ubd_t5;t5++) {
 for (__p=0; __p<nprocs; __p++) { receiver_list[__p] =
    ↪ 0; } sigma_y_1_0(2,t5,N,M, my_rank, nprocs,
    ↪ receiver_list); for (__p=0; __p<nprocs; __p++) {
    ↪ if (receiver_list[__p] != 0) { send_counts_y[__p]
    ↪ = pack_y_1_0(2,t5, send_buf_y[__p], send_counts_y[__p]);
    ↪ } };
}
IF_TIME(t_pack += rtclock() - t_pack_start);
if (N >= 1) {
 IF_TIME(t_comm_start = rtclock());
    ↪ MPI_Alltoall(send_counts_y, 1, MPI_INT,
    ↪ recv_counts_y, 1, MPI_INT, MPI_COMM_WORLD); req_count=0;
    ↪ for (__p=0; __p<nprocs; __p++) { if (send_counts_y[__p]
    ↪ >= 1) {IF_TIME(__total_count += send_counts_y[__p]);
    ↪ MPI_Isend(send_buf_y[__p], send_counts_y[__p], MPI_DOUBLE,
    ↪ __p, 123, MPI_COMM_WORLD, &reqs[req_count++]);}}for
```

```
    ↪ (__p=0; __p<nprocs; __p++) { if(recv_counts_y[__p]
    ↪ >= 1) { MPI_Irecv(recv_buf_y+displs_y[__p], recv_counts_y[__p],
    ↪ MPI_DOUBLE, __p, 123, MPI_COMM_WORLD, &reqs[req_count++]);}}
    ↪ MPI_Waitall(req_count, reqs, stats); for (__p=0;
    ↪ __p<nprocs; __p++) { send_counts_y[__p] = 0;}for
    ↪ (__p=0; __p<nprocs; __p++) { curr_displs_y[__p] =
    ↪ displs_y[__p]; }IF_TIME(t_comm += rtclock() - t_comm_start);;
}
IF_TIME(t_unpack_start = rtclock());
for (t5=0;t5<=N-1;t5++) {
 proc = pi_0(2,t5,N,M, nprocs); if ((my_rank != proc)
    ↪ && (recv_counts_y[proc] > 0)) { if
    ↪ (is_receiver_y_1_0(2,t5,N,M,my_rank,nprocs) != 0)
    ↪ { curr_displs_y[proc] = unpack_y_1_0(2,t5,recv_buf_y,curr_displs_y[proc]);
    ↪ }};
}
IF_TIME(t_unpack += rtclock() - t_unpack_start);
if ((M >= 1) && (N >= 1)) {
 for (t5=0;t5<=N+M-2;t5++) {
  IF_TIME(t_comp_start = rtclock());
_lb_dist=max(0,t5-M+1);
  _ub_dist=min(t5,N-1);
  polyrt_loop_dist(_lb_dist, _ub_dist, nprocs,
    ↪ my_rank, &lbd_t7, &ubd_t7);
  for (t7=lbd_t7;t7<=ubd_t7;t7++) {
   y[t7] = y[t7] + A[(t5-t7)][t7] * tmp[(t5-t7)];;
  }
IF_TIME(t_comp += rtclock() - t_comp_start);
  IF_TIME(t_pack_start = rtclock());
_lb_dist=max(0,t5-M+1);
  _ub_dist=min(t5,N-1);
  polyrt_loop_dist(_lb_dist, _ub_dist, nprocs,
    ↪ my_rank, &lbd_t7, &ubd_t7);
  for (t7=lbd_t7;t7<=ubd_t7;t7++) {
   for (__p=0; __p<nprocs; __p++) { receiver_list[__p]
    ↪ = 0; } sigma_y_1_3(3,t5,t7,N,M, my_rank,
    ↪ nprocs, receiver_list); for (__p=0; __p<nprocs; __p++)
    ↪ { if (receiver_list[__p] != 0) { send_counts_y[__p]
    ↪ = pack_y_1_3(3,t5,t7, send_buf_y[__p], send_counts_y[__p]);
    ↪ } };
  }
IF_TIME(t_pack += rtclock() - t_pack_start);
  IF_TIME(t_comm_start = rtclock());
    ↪ MPI_Alltoall(send_counts_y, 1, MPI_INT,
    ↪ recv_counts_y, 1, MPI_INT, MPI_COMM_WORLD); req_count=0;
    ↪ for (__p=0; __p<nprocs; __p++) { if (send_counts_y[__p]
    ↪ >= 1) {IF_TIME(__total_count += send_counts_y[__p]);
    ↪ MPI_Isend(send_buf_y[__p], send_counts_y[__p], MPI_DOUBLE,
    ↪ __p, 123, MPI_COMM_WORLD, &reqs[req_count++]);}}for
    ↪ (__p=0; __p<nprocs; __p++) { if(recv_counts_y[__p]
    ↪ >= 1) { MPI_Irecv(recv_buf_y+displs_y[__p], recv_counts_y[__p],
    ↪ MPI_DOUBLE, __p, 123, MPI_COMM_WORLD, &reqs[req_count++]);}}
    ↪ MPI_Waitall(req_count, reqs, stats); for (__p=0;
    ↪ __p<nprocs; __p++) { send_counts_y[__p] = 0;}for
    ↪ (__p=0; __p<nprocs; __p++) { curr_displs_y[__p] =
    ↪ displs_y[__p]; }IF_TIME(t_comm += rtclock() - t_comm_start);;
  IF_TIME(t_unpack_start = rtclock());
for (t7=max(0,t5-M+1);t7<=min(t5,N-1);t7++) {
   proc = pi_3(3,t5,t7,N,M, nprocs); if ((my_rank !=
    ↪ proc) && (recv_counts_y[proc] > 0)) { if
    ↪ (is_receiver_y_1_3(3,t5,t7,N,M,my_rank,nprocs) !=
```

```
    ↪ 0) { curr_displs_y[proc] = unpack_y_1_3(3,t5,t7,recv_buf_y,curr_displs_y[proc]);
    ↪ }};
  }
IF_TIME(t_unpack += rtclock() - t_unpack_start);
 }
}
/* End of CLooG code */
```

B.5 Статистика запусков atax (OpenMP)

Таблица 12 — Статистика запусков atax в вариантах vanilla, ilp_sync, p_ilp_sync: затраченное время, мс

#Нитей	Стат.	vanilla	ilp_sync	p_ilp_sync
1	μ	90.4116	126.5007	697.5997
	σ	5.361025	0.147984	0.693274
	Min	88.473	126.318	696.938
	Max	106.488	126.807	698.941
2	μ	–	94.6734	353.4006
	σ	–	1.090957	0.54426
	Min	–	93.802	352.67
	Max	–	96.905	354.419
4	μ	–	78.9874	177.9028
	σ	–	0.048732	0.200566
	Min	–	78.942	177.769
	Max	–	79.122	178.485
6	μ	–	72.2418	123.7199
	σ	–	0.02519	0.174394
	Min	–	72.212	123.538
	Max	–	72.293	124.132
8	μ	–	69.0015	98.2114
	σ	–	0.043926	0.121405
	Min	–	68.964	98.081
	Max	–	69.114	98.509

Таблица 13 — Статистика запусков atax в вариантах p_ilp_sync_mdp, pluto, p_pluto: затраченное время, мс

#Нитей	Стат.	p_ilp_sync_mdp	pluto	p_pluto
1	μ	98.6903	89.62	676.6468
	σ	0.606068	1.57972	1.249102
	Min	98.353	88.912	675.666
	Max	100.492	94.323	680.267
2	μ	57.4784	71.2693	335.5015
	σ	0.227202	0.061667	4.060652
	Min	57.216	71.18	333.071
	Max	57.922	71.421	347.061
4	μ	38.2125	66.4192	160.6508
	σ	0.078136	0.056482	0.175183

Продолжение таблицы 13

#Нитей	Стат.	p_ilp_sync_mdp	pluto	p_pluto
	Min	38.108	66.336	160.46
	Max	38.396	66.532	161.074
6	μ	31.756	64.3309	105.1241
	σ	0.136487	0.039157	1.110526
	Min	31.573	64.259	104.264
	Max	31.994	64.377	108.388
8	μ	30.5584	63.2884	80.0168
	σ	0.135678	0.025085	0.139926
	Min	30.402	63.254	79.869
	Max	30.791	63.34	80.321

B.6 Статистика запусков atax (MPI)

Таблица 14 — Статистика запусков atax_mpi в вариантах vanilla, p_ilp_arrays (_one, _two, _eq): затраченное время, мс

#Проц.	Стат.	vanilla	p_ilp_arrays_one	p_ilp_arrays_two	p_ilp_arrays_eq
1	μ	106.44311	671.05626	–	–
	σ	0.060561	18.414208	–	–
	Min	106.261	662.705	–	–
	Max	106.706	738.278	–	–
2	μ	–	622.55895	810.48944	–
	σ	–	2.285995	21.36778	–
	Min	–	620.07	799.437	–
	Max	–	637.025	879.101	–
4	μ	–	524.06098	677.83024	818.62033
	σ	–	0.708101	9.060428	14.781223
	Min	–	523.438	663.35	807.424
	Max	–	530.38	723.728	880.722
6	μ	–	404.58652	577.5592	793.97647
	σ	–	0.394439	5.998106	8.453014
	Min	–	404.012	572.986	787.317
	Max	–	407.345	615.548	850.707
8	μ	–	294.30056	518.00172	776.20827
	σ	–	1.296043	1.995882	10.340945
	Min	–	292.602	512.855	767.799
	Max	–	301.371	526.651	822.836

Таблица 15 — Статистика запусков atax_mpi в вариантах vanilla, p_ilp_arrays (_one, _two, _eq): доля времени вычислений, %

#Проц.	Стат.	vanilla	p_ilp_arrays_one	p_ilp_arrays_two	p_ilp_arrays_eq
1	μ	100.0	99.620489	–	–
	σ	0.0	0.007744	–	–
	Min	100.0	99.600957	–	–

Продолжение таблицы 15

#Проц.	Стат.	vanilla	p_ilp_arrays_one	p_ilp_arrays_two	p_ilp_arrays_eq
	Max	100.0	99.644636	–	–
2	μ	–	54.150673	40.590932	–
	σ	–	0.03541	0.15662	–
	Min	–	53.903745	40.071512	–
	Max	–	54.208536	41.291124	–
4	μ	–	33.478564	24.587381	19.612258
	σ	–	0.024354	0.087292	0.21893
	Min	–	33.285107	24.29288	19.015277
	Max	–	33.521389	25.221812	21.043996
6	μ	–	25.450449	17.696283	12.041178
	σ	–	0.023735	0.050119	0.245473
	Min	–	25.295618	17.476067	11.464585
	Max	–	25.498223	17.968312	13.348179
8	μ	–	24.0881	13.747837	8.894575
	σ	–	0.081194	0.043362	0.212649
	Min	–	23.740891	13.565153	8.431019
	Max	–	24.215978	13.938553	10.027484

Таблица 16 — Статистика запусков atax_mpi в вариантах p_ilp_arrays_mdp_sendrecv_distinp (_one, _two, _eq): затраченное время, мс

#Проц.	Стат.	p_ilp_arrays_mdp_ sendrecv_distinp_one	p_ilp_arrays_mdp_ sendrecv_distinp_two	p_ilp_arrays_mdp_ sendrecv_distinp_eq
1	μ	104.90684	–	–
	σ	5.049126	–	–
	Min	91.006	–	–
	Max	107.022	–	–
2	μ	126.49152	190.43703	–
	σ	5.253415	2.335185	–
	Min	122.888	184.531	–
	Max	137.164	193.711	–
4	μ	184.94022	226.35799	232.21546
	σ	5.34651	6.173231	0.82395
	Min	180.836	213.389	227.469
	Max	205.401	240.085	233.549
6	μ	186.86422	313.51822	258.10618
	σ	5.35657	0.965364	0.482483
	Min	182.447	308.907	256.782
	Max	194.972	316.179	259.092
8	μ	222.0823	384.20188	256.17829
	σ	0.761243	1.091125	0.764364
	Min	221.185	377.994	252.25
	Max	226.858	385.757	257.553

Таблица 17 — Статистика запусков atax_mpi в вариантах p_ilp_arrays_mdp_sendrecv_distinp (_one, _two, _eq): доля времени вычислений, %

#Проц.	Стат.	p_ilp_arrays_mdp_ sendrecv_distinp_one	p_ilp_arrays_mdp_ sendrecv_distinp_two	p_ilp_arrays_mdp_ sendrecv_distinp_eq
1	μ	99.348131	–	–
	σ	0.074598	–	–
	Min	99.00321	–	–

Продолжение таблицы 17

#Проц.	Стат.	p_ilp_arrays_mdp_ sendrecv_distinp_one	p_ilp_arrays_mdp_ sendrecv_distinp_two	p_ilp_arrays_mdp_ sendrecv_distinp_eq
	Max	99.610911	–	–
2	μ	40.26585	28.735434	–
	σ	0.32718	0.685277	–
	Min	38.987332	27.123432	–
	Max	40.908551	29.315799	–
4	μ	15.181414	11.774573	12.566489
	σ	0.201848	0.60534	0.045552
	Min	13.465842	11.144793	12.481776
	Max	15.438157	13.538778	12.816543
6	μ	12.115859	6.28693	8.185933
	σ	0.07606	0.050491	0.051621
	Min	11.750957	6.232538	7.973297
	Max	12.254699	6.505121	8.233891
8	μ	8.883471	4.255322	6.663217
	σ	0.020405	0.03107	0.058236
	Min	8.812792	4.22597	6.530475
	Max	8.939013	4.418377	6.81724

Таблица 18 — Статистика запусков atax_mpi в вариантах p_ilp_arrays_mdp_bcast_distinp (_one, _two, _eq): затраченное время, мс

#Проц.	Стат.	p_ilp_arrays_mdp_ bcast_distinp_one	p_ilp_arrays_mdp_ bcast_distinp_two	p_ilp_arrays_mdp_ bcast_distinp_eq
1	μ	103.4416	–	–
	σ	5.867482	–	–
	Min	90.297	–	–
	Max	106.318	–	–
2	μ	148.42976	307.96761	–
	σ	7.719371	2.037192	–
	Min	147.026	299.642	–
	Max	225.132	311.062	–
4	μ	165.92664	394.77706	617.36854
	σ	7.686196	25.660547	5.417344
	Min	156.196	377.113	605.009
	Max	236.089	627.381	635.988
6	μ	204.12611	413.05493	708.31023
	σ	10.398664	53.603981	5.837169
	Min	202.541	336.866	694.429
	Max	307.54	472.662	735.033
8	μ	252.72393	403.84032	1048.31529
	σ	15.690952	8.125157	6.292087
	Min	249.675	391.509	1031.746
	Max	405.494	421.11	1061.45

Таблица 19 — Статистика запусков atax_mpi в вариантах p_ilp_arrays_mdp_bcast_distinp (_one, _two, _eq): доля времени вычислений, %

#Проц.	Стат.	p_ilp_arrays_mdp_ bcast_distinp_one	p_ilp_arrays_mdp_ bcast_distinp_two	p_ilp_arrays_mdp_ bcast_distinp_eq
1	μ	99.529732	–	–
	σ	0.020298	–	–

Продолжение таблицы 19

#Проц.	Стат.	p_ilp_arrays_mdp_ bcast_distinp_one	p_ilp_arrays_mdp_ bcast_distinp_two	p_ilp_arrays_mdp_ bcast_distinp_eq
	Min	99.465033	–	–
	Max	99.57036	–	–
2	μ	33.272852	17.834117	–
	σ	0.866101	0.256927	–
	Min	24.709357	16.788319	–
	Max	33.653192	17.976275	–
4	μ	18.183522	6.79686	4.0137
	σ	0.623031	0.294103	0.059281
	Min	12.049337	4.174755	3.938442
	Max	18.330018	7.010023	4.230379
6	μ	10.751138	4.781966	2.44864
	σ	0.362227	0.657433	0.023025
	Min	7.158352	4.137452	2.384657
	Max	10.878801	5.763351	2.533637
8	μ	7.718328	3.991077	1.323119
	σ	0.293176	0.07355	0.007568
	Min	4.836168	3.868279	1.310176
	Max	7.803237	4.128687	1.345229

Таблица 20 — Статистика запусков atax_mpi в вариантах p_ilp_arrays_mdp_sendrecv_dupinp (_one, _two, _eq): затраченное время, мс

#Проц.	Стат.	p_ilp_arrays_mdp_ sendrecv_dupinp_one	p_ilp_arrays_mdp_ sendrecv_dupinp_two	p_ilp_arrays_mdp_ sendrecv_dupinp_eq
1	μ	105.84166	–	–
	σ	0.038562	–	–
	Min	105.753	–	–
	Max	105.931	–	–
2	μ	60.14942	69.9038	–
	σ	3.080233	0.087232	–
	Min	58.034	69.711	–
	Max	65.899	70.091	–
4	μ	37.1657	44.48694	57.11661
	σ	0.816578	1.410293	0.629786
	Min	36.69	43.189	56.261
	Max	39.488	49.716	59.016
6	μ	30.20295	36.19781	53.69989
	σ	0.153344	0.864283	0.363776
	Min	30.091	35.511	53.327
	Max	31.643	40.295	56.86
8	μ	27.84656	33.52534	52.61715
	σ	0.06913	0.817883	0.389818
	Min	27.705	32.907	52.168
	Max	28.124	38.381	56.127

Таблица 21 — Статистика запусков atax_mpi в вариантах p_ilp_arrays_mdp_sendrecv_dupinp (_one, _two, _eq): доля времени вычислений, %

#Проц.	Стат.	p_ilp_arrays_mdp_ sendrecv_dupinp_one	p_ilp_arrays_mdp_ sendrecv_dupinp_two	p_ilp_arrays_mdp_ sendrecv_dupinp_eq
1	μ	99.774271	–	–

Продолжение таблицы 21

#Проц.	Стат.	p_ilp_arrays_mdp_ sendrecv_dupinp_one	p_ilp_arrays_mdp_ sendrecv_dupinp_two	p_ilp_arrays_mdp_ sendrecv_dupinp_eq
	σ	0.013098	–	–
	Min	99.73485	–	–
	Max	99.806019	–	–
2	μ	84.494478	78.687484	–
	σ	0.165491	0.08344	–
	Min	84.106822	78.431047	–
	Max	84.813457	78.912563	–
4	μ	76.75428	58.860517	50.028907
	σ	0.220786	0.507766	0.491799
	Min	75.479166	56.050248	48.497167
	Max	77.067344	59.508366	50.710765
6	μ	72.079768	53.682002	38.174675
	σ	0.30932	0.375675	0.221642
	Min	69.420919	50.42503	36.359958
	Max	72.435979	54.118465	38.469673
8	μ	69.460557	47.485916	31.404862
	σ	0.180422	0.687704	0.31059
	Min	68.948426	41.373909	29.615303
	Max	69.861132	47.966886	31.704324

Таблица 22 — Статистика запусков atax_mpi в вариантах p_ilp_arrays_mdp_bcast_dupinp (_one, _two, _eq): затраченное время, мс

#Проц.	Стат.	p_ilp_arrays_mdp_ bcast_dupinp_one	p_ilp_arrays_mdp_ bcast_dupinp_two	p_ilp_arrays_mdp_ bcast_dupinp_eq
1	μ	105.809	–	–
	σ	0.057183	–	–
	Min	105.676	–	–
	Max	105.992	–	–
2	μ	58.50593	70.00335	–
	σ	0.518112	0.178471	–
	Min	57.942	69.783	–
	Max	60.434	71.585	–
4	μ	36.67307	46.84519	57.08923
	σ	0.089844	1.463752	0.636904
	Min	36.452	43.586	56.104
	Max	36.888	49.209	58.143
6	μ	30.15303	37.31189	53.64892
	σ	0.343601	0.578932	0.361823
	Min	29.942	35.464	53.091
	Max	31.739	37.887	56.589
8	μ	28.03965	33.69335	53.01316
	σ	0.081928	0.705432	0.241354
	Min	27.898	33.003	52.8
	Max	28.399	34.866	55.322

Таблица 23 — Статистика запусков atax_mpi в вариантах p_ilp_arrays_mdp_bcast_dupinp (_one, _two, _eq): доля времени вычислений, %

#Проц.	Стат.	p_ilp_arrays_mdp_bcast_dupinp_one	p_ilp_arrays_mdp_bcast_dupinp_two	p_ilp_arrays_mdp_bcast_dupinp_eq
1	μ	99.775386	–	–
	σ	0.014527	–	–
	Min	99.724722	–	–
	Max	99.809361	–	–
2	μ	84.504481	78.51729	–
	σ	0.088292	0.169793	–
	Min	84.264374	77.029436	–
	Max	84.723299	78.713835	–
4	μ	76.984022	58.425966	50.080249
	σ	0.166041	0.666416	0.501815
	Min	76.302824	57.398259	49.29767
	Max	77.309601	59.751935	50.898886
6	μ	72.863582	53.611519	38.102581
	σ	0.200589	0.395618	0.36263
	Min	72.230087	52.00831	36.246602
	Max	73.337109	54.246035	38.439066
8	μ	68.912587	47.54042	31.301814
	σ	0.194082	0.377479	0.141369
	Min	68.272463	45.732894	30.019558
	Max	69.260538	48.110867	31.423375

Таблица 24 — Статистика запусков atax_mpi в вариантах p_pluto (_one, _two, _eq): затраченное время, мс

#Проц.	Стат.	p_pluto_one	p_pluto_two	p_pluto_eq
1	μ	2356.841786	–	–
	σ	100.263043	–	–
	Min	2263.263226	–	–
	Max	2505.809784	–	–
2	μ	1695.207658	1817.373989	–
	σ	76.224166	24.576532	–
	Min	1583.637953	1686.064005	–
	Max	1772.286892	1846.518993	–
4	μ	1403.606887	1430.4987	1511.946251
	σ	28.308125	41.238146	21.769342
	Min	1308.308125	1302.140951	1419.440031
	Max	1443.63308	1481.481075	1562.700033
6	μ	1274.079857	1320.292294	1350.037272
	σ	35.101773	36.639235	18.082109
	Min	1190.099001	1222.020864	1310.314178
	Max	1320.179939	1378.894091	1389.539003
8	μ	1185.201044	1269.178228	1383.456595
	σ	36.238978	44.039335	25.818141
	Min	1150.341988	1210.123062	1349.076033
	Max	1254.110098	1340.617895	1485.707998

Таблица 25 — Статистика запусков atax_mpi в вариантах p_pluto (_one, _two, _eq): доля времени вычислений, %

#Проц.	Стат.	p_pluto_one	p_pluto_two	p_pluto_eq
1	μ	29.259619	–	–
	σ	0.222669	–	–
	Min	28.651797	–	–
	Max	29.728476	–	–
2	μ	19.00213	19.047428	–
	σ	1.931015	0.250851	–
	Min	15.634129	18.734058	–
	Max	21.287231	19.921547	–
4	μ	11.549903	10.721995	9.958104
	σ	0.141397	0.148653	0.149408
	Min	11.214545	10.472539	9.777382
	Max	11.823724	11.214844	10.341037
6	μ	7.9247	7.420821	4.979126
	σ	0.118128	0.135877	0.761211
	Min	7.707496	7.204306	4.214169
	Max	8.154172	7.772191	6.810074
8	μ	6.021338	5.540697	4.721468
	σ	0.064527	0.087141	0.273011
	Min	5.878191	5.407291	3.703446
	Max	6.220248	5.709538	5.05736

Таблица 26 — Разнородная нагрузка в запусках atax_mpi

Запуск	Затраченное время, мс			Доля затраченного времени, %		
vanilla: 1=1x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	106.44	0.00	0.00	100.00	0.00	0.00
p_ilp_arrays_one: 1=1x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	667.98	1.39	1.10	99.63	0.21	0.16
p_ilp_arrays_one: 2=1x2	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	622.32	103.72	65.59	72.79	16.67	10.54
Process 1	622.32	103.47	298.12	35.47	16.63	47.90
p_ilp_arrays_two: 2=2x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	808.42	274.61	87.55	55.20	33.97	10.83
Process 1	808.43	256.90	342.82	25.82	31.78	42.40
p_ilp_arrays_one: 4=1x4	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	524.00	156.85	134.76	44.35	29.93	25.72
Process 1	524.00	157.00	133.52	44.56	29.96	25.48
Process 2	524.00	156.85	133.52	44.59	29.93	25.48
Process 3	524.00	156.73	365.38	0.36	29.91	69.73
p_ilp_arrays_two: 4=2x2	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	676.65	299.55	151.29	33.37	44.27	22.36
Process 1	676.65	299.44	157.16	32.52	44.25	23.23
Process 2	676.65	287.75	170.83	32.23	42.53	25.25
Process 3	676.65	287.49	387.29	0.28	42.49	57.24
p_ilp_arrays_eq: 4=4x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	816.14	420.62	175.12	27.01	51.54	21.46
Process 1	816.14	419.60	187.03	25.67	51.41	22.92
Process 2	816.14	423.78	184.56	25.46	51.93	22.61
Process 3	816.14	412.21	402.01	0.23	50.51	49.26
p_ilp_arrays_one: 6=1x6	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	404.56	176.41	85.57	35.25	43.60	21.15
Process 1	404.56	176.36	89.59	34.26	43.59	22.14
Process 2	404.56	176.28	89.88	34.21	43.57	22.22
Process 3	404.56	176.21	90.23	34.14	43.56	22.30

Продолжение таблицы 26

Запуск	Затраченное время, мс			Доля затраченного времени, %		
Process 4	404.56	176.20	170.37	14.34	43.55	42.11
Process 5	404.56	176.04	226.54	0.49	43.51	56.00
p_ilp_arrays_two: 6=2x3	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	577.13	319.49	109.82	25.61	55.36	19.03
Process 1	577.13	319.36	120.64	23.76	55.34	20.90
Process 2	577.13	319.15	121.26	23.69	55.30	21.01
Process 3	577.13	307.54	138.17	22.77	53.29	23.94
Process 4	577.13	307.35	212.59	9.91	53.26	36.84
Process 5	577.13	307.43	267.78	0.33	53.27	46.40
p_ilp_arrays_eq: 6=6x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	792.98	491.14	161.80	17.66	61.94	20.40
Process 1	792.98	488.69	173.98	16.43	61.63	21.94
Process 2	792.98	495.36	167.55	16.40	62.47	21.13
Process 3	792.98	493.61	183.41	14.62	62.25	23.13
Process 4	792.98	491.57	244.65	7.16	61.99	30.85
Process 5	792.98	488.75	302.38	0.23	61.64	38.13
p_ilp_arrays_one: 8=1x8	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	294.26	124.41	70.06	33.91	42.28	23.81
Process 1	294.26	124.39	76.20	31.83	42.27	25.90
Process 2	294.26	124.48	76.18	31.81	42.30	25.89
Process 3	294.26	124.45	76.51	31.71	42.29	26.00
Process 4	294.26	124.50	77.14	31.48	42.31	26.21
Process 5	294.26	124.41	79.12	30.83	42.28	26.89
Process 6	294.26	124.39	167.96	0.65	42.27	57.08
Process 7	294.26	124.28	167.94	0.69	42.24	57.07
p_ilp_arrays_two: 8=2x4	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	517.84	318.96	95.23	20.02	61.59	18.39
Process 1	517.84	319.03	105.69	17.98	61.61	20.41
Process 2	517.84	319.06	106.06	17.91	61.61	20.48
Process 3	517.84	319.06	106.85	17.75	61.61	20.63
Process 4	517.84	308.73	115.68	18.04	59.62	22.34
Process 5	517.84	308.64	118.86	17.45	59.60	22.95
Process 6	517.84	308.63	207.28	0.37	59.60	40.03
Process 7	517.84	308.65	207.32	0.36	59.60	40.04
p_ilp_arrays_eq: 8=8x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	776.50	517.07	156.74	13.22	66.59	20.19
Process 1	776.50	514.92	167.32	12.14	66.31	21.55
Process 2	776.50	517.84	165.66	11.98	66.69	21.33
Process 3	776.50	515.03	179.51	10.55	66.33	23.12
Process 4	776.50	518.44	175.83	10.59	66.77	22.64
Process 5	776.49	517.86	168.25	11.64	66.69	21.67
Process 6	776.50	514.61	260.08	0.23	66.27	33.49
Process 7	776.50	510.41	264.10	0.26	65.73	34.01
p_ilp_arrays_mdp_ sendrecv_distinp_one: 1=1x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	106.61	0.47	0.26	99.32	0.44	0.24
p_ilp_arrays_mdp_ sendrecv_distinp_one: 2=1x2	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	126.11	66.55	1.91	45.72	52.77	1.51
Process 1	126.10	66.42	16.49	34.25	52.67	13.08
p_ilp_arrays_mdp_ sendrecv_distinp_two: 2=2x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	190.41	120.10	6.80	33.35	63.08	3.57
Process 1	190.41	120.03	22.45	25.17	63.04	11.79
p_ilp_arrays_mdp_ sendrecv_distinp_one: 4=1x4	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	185.78	110.44	43.15	17.33	59.45	23.22

Продолжение таблицы 26

Запуск	Затраченное время, мс			Доля затраченного времени, %		
Process 1	185.78	110.42	43.21	17.30	59.44	23.26
Process 2	185.78	110.45	43.27	17.26	59.45	23.29
Process 3	185.78	150.28	21.83	7.35	80.89	11.75
p_ilp_arrays_mdp_ sendrecv_distinp_two: 4=2x2	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	225.79	176.65	14.59	15.30	78.24	6.46
Process 1	225.79	176.47	14.89	15.25	78.15	6.59
Process 2	225.79	176.67	15.28	14.99	78.24	6.77
Process 3	225.79	176.63	33.68	6.86	78.23	14.92
p_ilp_arrays_mdp_ sendrecv_distinp_eq: 4=4x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	232.31	174.50	23.85	14.61	75.12	10.27
Process 1	232.30	174.50	23.83	14.62	75.12	10.26
Process 2	232.30	174.63	23.75	14.60	75.17	10.22
Process 3	232.30	174.78	42.62	6.42	75.24	18.35
p_ilp_arrays_mdp_ sendrecv_distinp_one: 6=1x6	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	187.58	157.21	4.89	13.59	83.81	2.61
Process 1	187.58	157.24	4.82	13.61	83.83	2.57
Process 2	187.58	157.28	4.99	13.49	83.85	2.66
Process 3	187.58	157.30	4.93	13.52	83.86	2.63
Process 4	187.58	157.46	9.92	10.77	83.94	5.29
Process 5	187.58	157.36	18.51	6.24	83.89	9.87
p_ilp_arrays_mdp_ sendrecv_distinp_two: 6=2x3	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	313.61	276.80	14.16	7.22	88.26	4.51
Process 1	313.61	276.72	14.04	7.29	88.24	4.48
Process 2	313.61	276.75	14.06	7.27	88.25	4.48
Process 3	313.62	276.55	14.80	7.10	88.18	4.72
Process 4	313.62	276.64	19.49	5.58	88.21	6.21
Process 5	313.62	276.80	26.74	3.22	88.26	8.53
p_ilp_arrays_mdp_ sendrecv_distinp_eq: 6=6x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	258.11	204.00	29.78	9.42	79.04	11.54
Process 1	258.11	204.01	29.77	9.43	79.04	11.53
Process 2	258.11	204.03	29.83	9.39	79.05	11.56
Process 3	258.11	203.99	29.88	9.39	79.03	11.57
Process 4	258.11	204.05	34.66	7.51	79.06	13.43
Process 5	258.11	204.21	43.49	4.03	79.12	16.85
p_ilp_arrays_mdp_ sendrecv_distinp_one: 8=1x8	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	222.00	194.03	5.49	10.12	87.40	2.47
Process 1	222.00	193.96	5.36	10.21	87.37	2.41
Process 2	222.00	193.99	5.27	10.24	87.39	2.37
Process 3	222.00	193.96	5.03	10.36	87.37	2.26
Process 4	222.00	194.00	5.25	10.25	87.39	2.36
Process 5	222.00	194.09	5.25	10.21	87.43	2.37
Process 6	222.00	194.10	17.04	4.89	87.43	7.68
Process 7	222.00	194.13	17.09	4.86	87.45	7.70
p_ilp_arrays_mdp_ sendrecv_distinp_two: 8=2x4	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	384.34	350.82	14.33	4.99	91.28	3.73
Process 1	384.34	350.85	14.24	5.01	91.29	3.71
Process 2	384.34	350.84	14.17	5.03	91.28	3.69
Process 3	384.34	350.88	14.29	4.99	91.29	3.72
Process 4	384.34	350.89	15.10	4.77	91.30	3.93
Process 5	384.34	350.93	15.14	4.75	91.31	3.94

Продолжение таблицы 26

Запуск	Затраченное время, мс			Доля затраченного времени, %		
Process 6	384.34	350.93	25.03	2.18	91.31	6.51
Process 7	384.34	350.95	24.95	2.20	91.31	6.49
p_ilp_arrays_mdp_ sendrecv_distinp_eq: 8=8x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	256.24	203.20	32.81	7.89	79.30	12.81
Process 1	256.24	203.29	32.87	7.83	79.34	12.83
Process 2	256.24	203.23	32.76	7.90	79.31	12.78
Process 3	256.24	203.39	32.90	7.79	79.37	12.84
Process 4	256.24	203.25	32.72	7.91	79.32	12.77
Process 5	256.24	203.28	32.67	7.92	79.33	12.75
Process 6	256.24	203.43	44.78	3.13	79.39	17.48
Process 7	256.24	203.39	44.75	3.16	79.37	17.46
p_ilp_arrays_mdp_ bcast_distinp_one: 1=1x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	106.10	0.25	0.22	99.55	0.24	0.21
p_ilp_arrays_mdp_ bcast_distinp_one: 2=1x2	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	147.65	89.18	1.85	38.35	60.40	1.25
Process 1	147.65	89.16	16.09	28.72	60.39	10.90
p_ilp_arrays_mdp_ bcast_distinp_two: 2=2x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	308.43	238.43	7.05	20.41	77.30	2.29
Process 1	308.43	238.39	22.63	15.37	77.29	7.34
p_ilp_arrays_mdp_ bcast_distinp_one: 4=1x4	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	166.16	126.84	4.44	21.00	76.33	2.67
Process 1	166.16	126.82	4.24	21.12	76.32	2.55
Process 2	166.16	126.80	4.28	21.11	76.31	2.57
Process 3	166.16	126.78	23.24	9.71	76.30	13.99
p_ilp_arrays_mdp_ bcast_distinp_two: 4=2x2	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	391.78	344.50	17.10	7.70	87.93	4.36
Process 1	391.78	344.57	17.14	7.68	87.95	4.38
Process 2	391.78	344.46	13.50	8.64	87.92	3.44
Process 3	391.78	344.55	32.12	3.86	87.94	8.20
p_ilp_arrays_mdp_ bcast_distinp_eq: 4=4x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	617.27	563.12	25.43	4.65	91.23	4.12
Process 1	617.27	563.11	25.75	4.60	91.23	4.17
Process 2	617.27	563.10	25.75	4.60	91.22	4.17
Process 3	617.27	563.12	41.49	2.05	91.23	6.72
p_ilp_arrays_mdp_ bcast_distinp_one: 6=1x6	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	203.08	172.83	5.37	12.25	85.10	2.65
Process 1	203.08	172.79	5.06	12.42	85.09	2.49
Process 2	203.08	172.79	5.23	12.34	85.09	2.58
Process 3	203.08	172.81	5.15	12.37	85.10	2.54
Process 4	203.08	172.84	10.17	9.88	85.11	5.01
Process 5	203.08	172.81	18.76	5.66	85.10	9.24
p_ilp_arrays_mdp_ bcast_distinp_two: 6=2x3	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	435.42	399.38	13.47	5.18	91.72	3.09
Process 1	435.42	399.51	13.51	5.15	91.75	3.10
Process 2	435.42	399.41	13.59	5.15	91.73	3.12
Process 3	435.43	399.42	14.23	5.00	91.73	3.27
Process 4	435.43	399.47	18.65	3.98	91.74	4.28
Process 5	435.43	399.48	26.14	2.25	91.74	6.00

Продолжение таблицы 26

Запуск	Затраченное время, мс			Доля затраченного времени, %		
p_ilp_arrays_mdp_ bcast_distinp_eq: 6=6x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	708.06	655.47	32.77	2.80	92.57	4.63
Process 1	708.07	655.45	32.76	2.80	92.57	4.63
Process 2	708.07	655.43	32.48	2.85	92.57	4.59
Process 3	708.07	655.42	32.77	2.81	92.56	4.63
Process 4	708.07	655.46	37.10	2.19	92.57	5.24
Process 5	708.07	655.42	43.89	1.24	92.56	6.20
p_ilp_arrays_mdp_ bcast_distinp_one: 8=1x8	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	250.78	222.77	5.82	8.85	88.83	2.32
Process 1	250.78	222.76	5.62	8.93	88.83	2.24
Process 2	250.78	222.77	5.36	9.03	88.83	2.14
Process 3	250.78	222.75	5.44	9.01	88.82	2.17
Process 4	250.78	222.76	5.75	8.88	88.83	2.29
Process 5	250.78	222.78	5.58	8.94	88.83	2.22
Process 6	250.78	222.74	17.48	4.21	88.82	6.97
Process 7	250.78	222.78	17.54	4.17	88.83	7.00
p_ilp_arrays_mdp_ bcast_distinp_two: 8=2x4	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	405.95	372.69	14.42	4.64	91.81	3.55
Process 1	405.95	372.77	14.11	4.70	91.83	3.48
Process 2	405.95	372.75	14.62	4.58	91.82	3.60
Process 3	405.95	372.78	14.26	4.66	91.83	3.51
Process 4	405.95	372.71	15.04	4.48	91.81	3.70
Process 5	405.95	372.76	14.91	4.50	91.82	3.67
Process 6	405.95	372.74	25.02	2.02	91.82	6.16
Process 7	405.95	372.75	25.02	2.02	91.82	6.16
p_ilp_arrays_mdp_ bcast_distinp_eq: 8=8x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	1048.32	989.43	42.52	1.56	94.38	4.06
Process 1	1048.32	989.44	42.87	1.53	94.38	4.09
Process 2	1048.32	989.40	42.83	1.53	94.38	4.09
Process 3	1048.32	989.40	42.77	1.54	94.38	4.08
Process 4	1048.32	989.42	42.61	1.55	94.38	4.06
Process 5	1048.31	989.39	42.62	1.56	94.38	4.07
Process 6	1048.32	989.38	52.07	0.66	94.38	4.97
Process 7	1048.32	989.38	52.10	0.65	94.38	4.97
p_ilp_arrays_mdp_ sendrecv_dupinp_one: 1=1x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	105.84	0.00	0.24	99.78	0.00	0.22
p_ilp_arrays_mdp_ sendrecv_dupinp_one: 2=1x2	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	60.55	0.04	1.92	96.76	0.07	3.17
Process 1	60.55	0.04	16.66	72.42	0.07	27.51
p_ilp_arrays_mdp_ sendrecv_dupinp_two: 2=2x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	69.90	0.05	7.03	89.88	0.07	10.06
Process 1	69.90	0.04	22.72	67.45	0.06	32.49
p_ilp_arrays_mdp_ sendrecv_dupinp_one: 4=1x4	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	37.04	0.08	4.29	88.18	0.23	11.59
Process 1	37.04	0.08	4.13	88.62	0.23	11.15
Process 2	37.04	0.08	4.08	88.76	0.23	11.01
Process 3	37.04	0.04	21.84	40.92	0.12	58.96
p_ilp_arrays_mdp_ sendrecv_dupinp_two: 4=2x2	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация

Продолжение таблицы 26

Запуск	Затраченное время, мс			Доля затраченного времени, %		
Process 0	44.44	0.06	13.72	68.98	0.13	30.89
Process 1	44.44	0.06	13.78	68.86	0.13	31.01
Process 2	44.44	0.06	14.68	66.82	0.14	33.04
Process 3	44.44	0.04	31.26	29.58	0.09	70.33
p_ilp_arrays_mdp_ sendrecv_dupinp_eq: 4=4x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	57.09	0.06	24.02	57.82	0.10	42.08
Process 1	57.10	0.06	23.83	58.17	0.10	41.73
Process 2	57.10	0.06	23.67	58.43	0.11	41.46
Process 3	57.10	0.04	42.50	25.48	0.08	74.44
p_ilp_arrays_mdp_ sendrecv_dupinp_one: 6=1x6	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	30.19	0.07	5.59	81.26	0.24	18.50
Process 1	30.19	0.07	5.03	83.08	0.24	16.68
Process 2	30.19	0.07	5.12	82.79	0.24	16.97
Process 3	30.19	0.07	5.09	82.88	0.24	16.88
Process 4	30.19	0.07	10.42	65.23	0.24	34.53
Process 5	30.19	0.03	18.73	37.85	0.10	62.05
p_ilp_arrays_mdp_ sendrecv_dupinp_two: 6=2x3	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	36.18	0.08	13.53	62.39	0.22	37.39
Process 1	36.18	0.08	13.50	62.45	0.22	37.33
Process 2	36.18	0.08	13.66	62.02	0.22	37.76
Process 3	36.18	0.12	14.36	59.99	0.33	39.69
Process 4	36.18	0.11	18.82	47.67	0.31	52.02
Process 5	36.18	0.05	26.41	26.86	0.14	73.00
p_ilp_arrays_mdp_ sendrecv_dupinp_eq: 6=6x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	53.67	0.07	30.25	43.50	0.13	56.37
Process 1	53.67	0.07	30.02	43.94	0.12	55.94
Process 2	53.67	0.08	29.91	44.13	0.14	55.73
Process 3	53.66	0.07	29.92	44.12	0.13	55.75
Process 4	53.66	0.07	34.73	35.14	0.14	64.72
Process 5	53.67	0.05	43.74	18.40	0.09	81.51
p_ilp_arrays_mdp_ sendrecv_dupinp_one: 8=1x8	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	27.84	0.07	5.69	79.31	0.27	20.43
Process 1	27.84	0.08	5.46	80.09	0.29	19.62
Process 2	27.84	0.07	5.09	81.45	0.27	18.28
Process 3	27.84	0.08	5.41	80.27	0.29	19.44
Process 4	27.84	0.07	5.49	80.00	0.27	19.73
Process 5	27.84	0.08	5.70	79.23	0.29	20.47
Process 6	27.84	0.07	17.20	37.99	0.24	61.76
Process 7	27.84	0.07	17.29	37.65	0.24	62.11
p_ilp_arrays_mdp_ sendrecv_dupinp_two: 8=2x4	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	33.45	0.08	14.89	55.24	0.25	44.51
Process 1	33.45	0.08	14.44	56.58	0.24	43.18
Process 2	33.45	0.08	14.67	55.88	0.25	43.86
Process 3	33.45	0.09	14.81	55.45	0.26	44.28
Process 4	33.45	0.09	15.44	53.58	0.28	46.14
Process 5	33.45	0.10	15.39	53.70	0.30	45.99
Process 6	33.45	0.07	25.31	24.16	0.20	75.64
Process 7	33.45	0.07	25.13	24.68	0.20	75.12
p_ilp_arrays_mdp_ sendrecv_dupinp_eq: 8=8x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	52.58	0.06	33.20	36.76	0.11	63.13

Продолжение таблицы 26

Запуск	Затраченное время, мс			Доля затраченного времени, %		
Process 1	52.59	0.07	32.99	37.14	0.13	62.73
Process 2	52.58	0.06	32.93	37.26	0.12	62.63
Process 3	52.59	0.07	32.88	37.34	0.13	62.54
Process 4	52.58	0.07	32.87	37.36	0.13	62.51
Process 5	52.59	0.07	32.76	37.56	0.14	62.30
Process 6	52.58	0.05	44.94	14.45	0.09	85.46
Process 7	52.59	0.05	44.95	14.44	0.09	85.47
p_ilp_arrays_mdp_ bcast_dupinp_one: 1=1x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	105.81	0.00	0.23	99.79	0.00	0.21
p_ilp_arrays_mdp_ bcast_dupinp_one: 2=1x2	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	58.48	0.06	2.02	96.44	0.11	3.46
Process 1	58.48	0.07	15.91	72.68	0.11	27.21
p_ilp_arrays_mdp_ bcast_dupinp_two: 2=2x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	69.99	0.07	7.14	89.70	0.10	10.20
Process 1	69.99	0.06	22.68	67.52	0.08	32.40
p_ilp_arrays_mdp_ bcast_dupinp_one: 4=1x4	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	36.67	0.07	4.21	88.32	0.19	11.49
Process 1	36.68	0.07	3.95	89.04	0.19	10.77
Process 2	36.67	0.07	3.96	89.00	0.19	10.81
Process 3	36.67	0.07	21.45	41.32	0.19	58.48
p_ilp_arrays_mdp_ bcast_dupinp_two: 4=2x2	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	46.84	0.11	16.53	64.47	0.23	35.30
Process 1	46.84	0.11	16.29	64.98	0.23	34.78
Process 2	46.84	0.09	13.42	71.17	0.19	28.65
Process 3	46.84	0.08	31.87	31.78	0.18	68.05
p_ilp_arrays_mdp_ bcast_dupinp_eq: 4=4x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	57.28	0.14	24.08	57.72	0.24	42.04
Process 1	57.28	0.15	23.81	58.17	0.27	41.56
Process 2	57.28	0.15	23.86	58.08	0.27	41.65
Process 3	57.27	0.13	42.58	25.44	0.22	74.34
p_ilp_arrays_mdp_ bcast_dupinp_one: 6=1x6	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	30.07	0.07	5.39	81.87	0.22	17.91
Process 1	30.07	0.07	4.76	83.93	0.23	15.84
Process 2	30.07	0.07	4.84	83.68	0.23	16.09
Process 3	30.07	0.07	4.97	83.27	0.22	16.51
Process 4	30.07	0.07	9.98	66.58	0.22	33.20
Process 5	30.07	0.07	18.60	37.94	0.22	61.84
p_ilp_arrays_mdp_ bcast_dupinp_two: 6=2x3	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	37.38	0.13	13.11	64.59	0.34	35.07
Process 1	37.38	0.13	12.98	64.93	0.34	34.72
Process 2	37.38	0.13	13.11	64.59	0.34	35.06
Process 3	37.38	0.12	15.51	58.20	0.32	41.48
Process 4	37.38	0.11	20.15	45.81	0.30	53.89
Process 5	37.38	0.11	27.55	26.02	0.29	73.69
p_ilp_arrays_mdp_ bcast_dupinp_eq: 6=6x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	53.62	0.15	30.03	43.71	0.28	56.01
Process 1	53.62	0.16	29.90	43.95	0.29	55.76
Process 2	53.62	0.16	29.90	43.94	0.30	55.76

Продолжение таблицы 26

Запуск	Затраченное время, мс			Доля затраченного времени, %		
Process 3	53.62	0.16	29.84	44.04	0.30	55.66
Process 4	53.62	0.16	34.76	34.88	0.30	64.82
Process 5	53.62	0.14	43.57	18.49	0.26	81.25
p_ilp_arrays_mdp_ bcast_dupinp_one: 8=1x8	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	28.03	0.08	5.67	79.48	0.29	20.24
Process 1	28.03	0.08	5.71	79.36	0.29	20.36
Process 2	28.03	0.08	5.63	79.62	0.29	20.09
Process 3	28.03	0.08	5.55	79.92	0.28	19.80
Process 4	28.03	0.08	5.67	79.49	0.29	20.23
Process 5	28.03	0.08	5.81	79.00	0.28	20.72
Process 6	28.03	0.08	17.38	37.68	0.30	62.02
Process 7	28.03	0.08	17.54	37.13	0.29	62.58
p_ilp_arrays_mdp_ bcast_dupinp_two: 8=2x4	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	33.61	0.08	15.34	54.12	0.23	45.65
Process 1	33.61	0.08	14.10	57.80	0.24	41.96
Process 2	33.61	0.08	14.80	55.73	0.24	44.03
Process 3	33.61	0.08	14.91	55.38	0.24	44.37
Process 4	33.61	0.07	15.63	53.28	0.21	46.51
Process 5	33.61	0.07	15.42	53.91	0.21	45.88
Process 6	33.61	0.07	25.50	23.92	0.21	75.87
Process 7	33.61	0.06	25.48	24.00	0.19	75.81
p_ilp_arrays_mdp_ bcast_dupinp_eq: 8=8x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	52.99	0.22	33.25	36.84	0.42	62.74
Process 1	52.99	0.22	33.12	37.08	0.42	62.51
Process 2	52.99	0.21	33.00	37.32	0.40	62.28
Process 3	52.99	0.21	33.30	36.77	0.39	62.84
Process 4	52.99	0.22	32.96	37.40	0.41	62.19
Process 5	52.99	0.21	33.22	36.91	0.39	62.70
Process 6	52.98	0.21	45.18	14.34	0.39	85.27
Process 7	52.99	0.20	45.22	14.29	0.37	85.34
p_pluto_one: 1=1x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	2332.84	1664.44	0.00	28.65	71.35	0.00
p_pluto_one: 2=1x2	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	1695.96	1346.91	0.00	20.58	79.42	0.00
Process 1	1695.96	1496.09	0.00	11.78	88.22	0.00
p_pluto_two: 2=2x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	1818.83	1448.53	0.00	20.36	79.64	0.00
Process 1	1818.83	1501.34	0.00	17.46	82.54	0.00
p_pluto_one: 4=1x4	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	1406.67	1244.44	0.00	11.53	88.47	0.00
Process 1	1406.67	1245.08	0.00	11.49	88.51	0.00
Process 2	1406.67	1245.40	0.00	11.46	88.54	0.00
Process 3	1406.67	1244.09	0.00	11.56	88.44	0.00
p_pluto_two: 4=2x2	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	1434.34	1265.29	0.00	11.79	88.21	0.00
Process 1	1434.34	1264.99	0.00	11.81	88.19	0.00
Process 2	1434.34	1291.82	0.00	9.94	90.06	0.00
Process 3	1434.34	1292.91	0.00	9.86	90.14	0.00
p_pluto_eq: 4=4x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	1512.86	1373.39	0.00	9.22	90.78	0.00
Process 1	1512.86	1371.30	0.00	9.36	90.64	0.00
Process 2	1512.86	1340.54	0.00	11.39	88.61	0.00
Process 3	1512.86	1370.90	0.00	9.38	90.62	0.00
p_pluto_one: 6=1x6	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация

Продолжение таблицы 26

Запуск	Затраченное время, мс			Доля затраченного времени, %		
Process 0	1274.09	1172.43	0.00	7.98	92.02	0.00
Process 1	1274.09	1171.88	0.00	8.02	91.98	0.00
Process 2	1274.09	1173.85	0.00	7.87	92.13	0.00
Process 3	1274.09	1174.11	0.00	7.85	92.15	0.00
Process 4	1274.09	1175.01	0.00	7.78	92.22	0.00
Process 5	1274.09	1173.86	0.00	7.87	92.13	0.00
p_pluto_two: 6=2x3	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	1320.34	1211.23	0.00	8.26	91.74	0.00
Process 1	1320.34	1210.41	0.00	8.33	91.67	0.00
Process 2	1320.34	1213.25	0.00	8.11	91.89	0.00
Process 3	1320.34	1230.18	0.00	6.83	93.17	0.00
Process 4	1320.34	1228.08	0.00	6.99	93.01	0.00
Process 5	1320.34	1230.31	0.00	6.82	93.18	0.00
p_pluto_eq: 6=6x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	1350.03	1265.72	0.00	6.25	93.75	0.00
Process 1	1350.03	1296.97	0.00	3.93	96.07	0.00
Process 2	1350.04	1296.51	0.00	3.96	96.04	0.00
Process 3	1350.03	1296.36	0.00	3.98	96.02	0.00
Process 4	1350.03	1275.66	0.00	5.51	94.49	0.00
Process 5	1350.03	1296.55	0.00	3.96	96.04	0.00
p_pluto_one: 8=1x8	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	1185.10	1115.43	0.00	5.88	94.12	0.00
Process 1	1185.10	1114.69	0.00	5.94	94.06	0.00
Process 2	1185.10	1115.20	0.00	5.90	94.10	0.00
Process 3	1185.10	1115.44	0.00	5.88	94.12	0.00
Process 4	1185.10	1115.66	0.00	5.86	94.14	0.00
Process 5	1185.10	1115.61	0.00	5.86	94.14	0.00
Process 6	1185.10	1115.12	0.00	5.90	94.10	0.00
Process 7	1185.10	1115.12	0.00	5.90	94.10	0.00
p_pluto_two: 8=2x4	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	1267.07	1190.25	0.00	6.06	93.94	0.00
Process 1	1267.07	1190.17	0.00	6.07	93.93	0.00
Process 2	1267.07	1190.90	0.00	6.01	93.99	0.00
Process 3	1267.07	1191.88	0.00	5.93	94.07	0.00
Process 4	1267.07	1199.37	0.00	5.34	94.66	0.00
Process 5	1267.07	1199.90	0.00	5.30	94.70	0.00
Process 6	1267.06	1200.45	0.00	5.26	94.74	0.00
Process 7	1267.07	1200.87	0.00	5.22	94.78	0.00
p_pluto_eq: 8=8x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	1377.78	1313.24	0.00	4.68	95.32	0.00
Process 1	1377.79	1315.23	0.00	4.54	95.46	0.00
Process 2	1377.79	1326.12	0.00	3.75	96.25	0.00
Process 3	1377.79	1312.72	0.00	4.72	95.28	0.00
Process 4	1377.78	1313.17	0.00	4.69	95.31	0.00
Process 5	1377.79	1312.45	0.00	4.74	95.26	0.00
Process 6	1377.79	1319.33	0.00	4.24	95.76	0.00
Process 7	1377.79	1323.44	0.00	3.94	96.06	0.00

Приложение Г

Распараллеливание программы syr2k на языке C

Г.1 Описание программы

Процедура syr2k из пакета BLAS подразумевает вычисление $C_{out} = \alpha AB^T + \alpha BA^T + \beta C$, где α и β — скаляры, A и B — матрицы размера $N \times M$, C и C_{out} — симметричные матрицы размера $N \times N$.

Листинг Г.1 Процедура syr2k (последовательный вариант) на языке C

```
1  void syr2k_vanilla(int N, int M, double** A, double** B, double** C, double alpha, double beta) {
2  #pragma scop
3    for (int i = 0; i < N; i++) {
4      for (int j = 0; j <= i; j++)
5        C[i][j] *= beta; //S0
6      for (int k = 0; k < M; k++)
7        for (int j = 0; j <= i; j++)
8          C[i][j] += alpha * (A[j][k] * B[i][k] + B[j][k] * A[i][k]); //S1
9    }
10 #pragma endscop
11 }
```

На рисунке Г.1 проиллюстрирован обобщенный граф зависимостей фрагмента функции `syr2k_vanilla`. Зависимости R1, R5 имеют тип «чтение после записи», зависимости R0, R3, R4 имеют тип «запись после чтения», зависимости R2, R6 имеют тип «запись после записи».

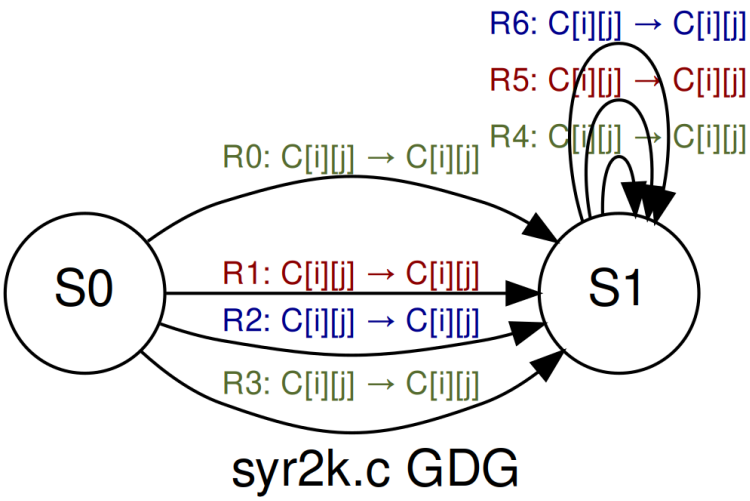


Рисунок Г.1 — Обобщенный граф зависимостей syr2k_vanilla

Г.2 Журнал трансляции ilpy

Листинг Г.2 Журнал трансляции ilpy при нахождении аффинных отображений для syr2k

```
#Расписание вычислений

R6 (weight 499950): L=1
R5 (weight 499950): L=1
R4 (weight 499950): L=1
R3 (weight 505000): L=M
R2 (weight 505000): L=M
R1 (weight 505000): L=M
R0 (weight 505000): L=M
zero: 0, constant: 3, affine: 4, total: 7
C = 203499850.000000

#Размещение вычислений

# component 1 with linear independence ENABLED and FCO property ENABLED
R6 (weight 499950): L=0
R5 (weight 499950): L=0
R4 (weight 499950): L=0
R3 (weight 505000): L=0
R2 (weight 505000): L=0
R1 (weight 505000): L=0
R0 (weight 505000): L=0
zero: 7, constant: 0, affine: 0, total: 7
C = 0.000000
```

Листинг Г.3 Журнал трансляции ilpy при нахождении аффинных отображений для syr2k с параметром `--async`

```
#Размещение вычислений, первый компонент

# component 1 with linear independence ENABLED and FCO property ENABLED
R6 (weight 499950): L=0
R5 (weight 499950): L=0
R4 (weight 499950): L=0
R3 (weight 505000): L=0
R2 (weight 505000): L=0
R1 (weight 505000): L=0
R0 (weight 505000): L=0
zero: 7, constant: 0, affine: 0, total: 7
C = 0.000000

#Размещение вычислений, второй компонент

# component 2 with linear independence ENABLED and FCO property ENABLED
R6 (weight 499950): L=0
R5 (weight 499950): L=0
R4 (weight 499950): L=0
R3 (weight 505000): L=0
R2 (weight 505000): L=0
R1 (weight 505000): L=0
R0 (weight 505000): L=0
```

```
zero: 7, constant: 0, affine: 0, total: 7
C = 0.000000
```

Листинг Г.4 Журнал трансляции ilpy при нахождении размещения вычислений и данных для syr2k

```
#Первый компонент

# component 1 with linear independence ENABLED and FCO property ENABLED
S0, A2 (READ C[i][j], weight 5050): L=0
S0, A1 (WRITE C[i][j], weight 5050): L=0
S0, A0 (READ C[i][j], weight 5050): L=0
S1, A5 (READ A[i][k], weight 505000): L=0
S1, A4 (READ B[j][k], weight 505000): L=N
S1, A3 (READ B[i][k], weight 505000): L=0
S1, A2 (READ A[j][k], weight 505000): L=N
S1, A1 (WRITE C[i][j], weight 505000): L=0
S1, A0 (READ C[i][j], weight 505000): L=0
zero: 7, constant: 0, affine: 2, total: 9
C = 101000000.000000

#Второй компонент

# component 2 with linear independence ENABLED and FCO property ENABLED
S0, A2 (READ C[i][j], weight 5050): L=0
S0, A1 (WRITE C[i][j], weight 5050): L=0
S0, A0 (READ C[i][j], weight 5050): L=0
S1, A5 (READ A[i][k], weight 505000): L=N
S1, A4 (READ B[j][k], weight 505000): L=0
S1, A3 (READ B[i][k], weight 505000): L=N
S1, A2 (READ A[j][k], weight 505000): L=0
S1, A1 (WRITE C[i][j], weight 505000): L=0
S1, A0 (READ C[i][j], weight 505000): L=0
zero: 7, constant: 0, affine: 2, total: 9
C = 101000000.000000
```

Г.3 Результат работы ilpy

Листинг Г.5 syr2k (параллельный вариант ilpy без директив OpenMP, асинхронный параллелизм) на языке C

```
if (N >= 1) {
 for (ilpp=0;ilpp<=N-1;ilpp++) {
  for (p1=-4*ilpp+12*N-12;p1<=-3*ilpp+12*N-12;p1++) {
   C[ilpp][4*ilpp+p1-12*N+12] += C[ilpp][4*ilpp+p1-12*N+12];
   for (t0=1;t0<=M;t0++) {
    C[ilpp][4*ilpp+p1-12*N+12] += A[4*ilpp+p1-12*N+12][t0-1]*B[ilpp][t0-1] +
    ↪ B[4*ilpp+p1-12*N+12][t0-1]*A[ilpp][t0-1];
   }
  }
```

```
 }
}
```

Листинг Г.6 syr2k_ilp_async (параллельный вариант ilpy с директивами OpenMP, асинхронный параллелизм) на языке C

```
void syr2k_ilp_async(int N, int M, double** A, double** B, double** C, double alpha, double beta) {
  if (N >= 1) {
    #pragma omp parallel for
    for (int ilpp=0;ilpp<=N-1;ilpp++) {
      for (int p1=-4*ilpp+12*N-12;p1<=-3*ilpp+12*N-12;p1++) {
        C[ilpp][4*ilpp+p1-12*N+12] *= beta;
        for (int t0=1;t0<=M;t0++) {
          C[ilpp][4*ilpp+p1-12*N+12] += alpha*(A[4*ilpp+p1-12*N+12][t0-1]*B[ilpp][t0-1] +
    ↪ B[4*ilpp+p1-12*N+12][t0-1]*A[ilpp][t0-1]);
        }
      }
    }
  }
}
```

Листинг Г.7 syr2k (параллельный вариант ilpy без конструкций MPI, асинхронный параллелизм) на языке C

```
if (N >= 1) {
 for (ilpp=0;ilpp<=N-1;ilpp++) {
  for (p1=-ilpp+N-1;p1<=N-1;p1++) {
   C[ilpp][-p1+N-1] += C[ilpp][-p1+N-1];
   for (t0=1;t0<=M;t0++) {
    C[ilpp][-p1+N-1] += A[-p1+N-1][t0-1]*B[ilpp][t0-1] + B[-p1+N-1][t0-1]*A[ilpp][t0-1];
   }
  }
 }
}
```

Листинг Г.8 syr2k_ilp_arrays_async (параллельный вариант ilpy с конструкциями MPI, асинхронный параллелизм) на языке C

```
// arrays placement

#define T_base 1000000

#define eta_A(i0,i1) (i0)
#define T_A (1 * T_base)

int A_chunk_size;
int A_actual_chunk_size;

#define eta_B(i0,i1) (i0)
#define T_B (2 * T_base)

int B_chunk_size;
int B_actual_chunk_size;

#define eta_C(i0,i1) (i0)
#define T_C (3 * T_base)
```

```
int C_chunk_size;
int C_actual_chunk_size;

// mpi routine

#define eta_A_i(i1) eta_A(i,i1)
#define eta_B_i(i1) eta_B(i,i1)

void syr2k_ilp_arrays_async(int N, int M, double** A, double** B, double** C, double alpha, double
    ↪ beta) {
  if (N >= 1) {
    // A[-p1+N-1][t0-1] in C[ilpp][-p1+N-1] += alpha * (A[-p1+N-1][t0-1]*B[ilpp][t0-1] +
    ↪ B[-p1+N-1][t0-1]*A[ilpp][t0-1])
    for (int i=0;i<=N-1;i++) {
      line_Q_read_send(0, clamp(lr, 0, N-1), M-1, clamp(ur, 0, N-1), A[i], eta_A_i, T_A + i);
      line_Q_read_receive(0, clamp(lR, 0, N-1), M-1, clamp(uR, 0, N-1), A[i], eta_A_i, T_A + i);
    }
    //
    // B[-p1+N-1][t0-1] in C[ilpp][-p1+N-1] += alpha * (A[-p1+N-1][t0-1]*B[ilpp][t0-1] +
    ↪ B[-p1+N-1][t0-1]*A[ilpp][t0-1])
    for (int i=0;i<=N-1;i++) {
      line_Q_read_send(0, clamp(lr, 0, N-1), M-1, clamp(ur, 0, N-1), B[i], eta_B_i, T_B + i);
      line_Q_read_receive(0, clamp(lR, 0, N-1), M-1, clamp(uR, 0, N-1), B[i], eta_B_i, T_B + i);
    }
    //
    for (int ilpp=max(lR,0);ilpp<=min(uR,N-1);ilpp++) {
      for (int p1=-ilpp+N-1;p1<=N-1;p1++) {
        C[ilpp][-p1+N-1] *= beta;
        for (int t0=1;t0<=M;t0++) {
          C[ilpp][-p1+N-1] += alpha * (A[-p1+N-1][t0-1]*B[ilpp][t0-1] +
    ↪ B[-p1+N-1][t0-1]*A[ilpp][t0-1]);
        }
      }
    }
    MPI_Barrier_time(MPI_COMM_WORLD);
    collect_matrix_rows_double(C, N, N, C_chunk_size);
  }
}
```

Г.4 Преобразования pluto

Листинг Г.9 Преобразования pluto в формате cloog для syr2k_pluto

```
# Number of scattering functions
2

# T(S1)
4 10
0 1 0 0 0 0 0 0 0 0
0 0 1 0 0 -1 0 0 0 0
0 0 0 1 0 0 -1 0 0 0
0 0 0 0 1 0 0 0 0 0
```

```
# T(S2)
4 11
0 1 0 0 0 0 0 0 0 0 -1
0 0 1 0 0 -1 0 0 0 0 0
0 0 0 1 0 0 0 -1 0 0 0
0 0 0 0 1 0 -1 0 0 0 0
```

Листинг Г.10 syr2k (параллельный вариант pluto с директивами OpenMP, асинхронный параллелизм) на языке C

```
 int t1, t2, t3, t4;
 int lb, ub, lbd, ubd, lb2, ub2;
 register int lbv, ubv;
/* Start of CLooG code */
if (N >= 1) {
 {/* extra braces to avoid redefination of lbp*/
int lbp=0;
 int ubp=N-1;
#pragma omp parallel for private(lbv,ubv,t3,t4)
 for (t2=lbp;t2<=ubp;t2++) {
  lbv=0;
  ubv=t2;
#pragma ivdep
#pragma vector always
  for (t3=lbv;t3<=ubv;t3++) {
   C[t2][t3] += C[t2][t3];;
  }
 }
}/*end of omp parallel loop */
 if (M >= 1) {
  {/* extra braces to avoid redefination of lbp*/
int lbp=0;
  int ubp=N-1;
#pragma omp parallel for private(lbv,ubv,t3,t4)
  for (t2=lbp;t2<=ubp;t2++) {
   for (t3=0;t3<=t2;t3++) {
    for (t4=0;t4<=M-1;t4++) {
     C[t2][t3] += A[t3][t4]*B[t2][t4] + B[t3][t4]*A[t2][t4];;
    }
   }
  }
}/*end of omp parallel loop */
 }
}
/* End of CLooG code */
```

Листинг Г.11 syr2k_pluto (обработанный вариант pluto с директивами OpenMP, асинхронный параллелизм) на языке C

```
void syr2k_pluto(int N, int M, double** A, double** B, double** C, double alpha, double beta) {
  #pragma omp parallel
  {
    /* Start of CLooG code */
    if (N >= 1) {
      #pragma omp for
      for (int t2=0;t2<=N-1;t2++) {
        for (int t3=0;t3<=t2;t3++) {
          C[t2][t3] *= beta;
```

```
        }
      }
      if (M >= 1) {
        #pragma omp for
        for (int t2=0;t2<=N-1;t2++) {
          for (int t3=0;t3<=t2;t3++) {
            for (int t4=0;t4<=M-1;t4++) {
              C[t2][t3] += alpha*(A[t3][t4]*B[t2][t4] + B[t3][t4]*A[t2][t4]);
            }
          }
        }
      }
    }
    /* End of CLooG code */
  }
}
```

Листинг Г.12 Преобразования pluto в формате cloog для syr2k_pluto_mpi

```
# Number of scattering functions
8

# T(S1)
6 13
0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 -1 0 0 0 0 0
0 0 0 0 0 1 0 0 -1 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0

# T(S2)
6 14
0 1 0 0 0 0 0 0 0 0 0 0 0 -1
0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 -1 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 -1 0 0 0 0
0 0 0 0 0 0 1 0 -1 0 0 0 0 0
```

Листинг Г.13 syr2k_pluto_mpi (параллельный вариант pluto с конструкциями MPI, асинхронный параллелизм) на языке C

```
/* Start of CLooG code */
if (N >= 1) {
 IF_TIME(t_comp_start = rtclock());
_lb_dist=0;
 _ub_dist=N-1;
 polyrt_loop_dist(_lb_dist, _ub_dist, nprocs, my_rank,
    ↪ &lbd_t4, &ubd_t4);
 for (t4=lbd_t4;t4<=ubd_t4;t4++) {
  lbv=0;
  ubv=t4;
#pragma ivdep
#pragma vector always
  for (t5=lbv;t5<=ubv;t5++) {
   C[t4][t5] *= beta;;
  }
 }
```

```
IF_TIME(t_comp += rtclock() - t_comp_start);
 IF_TIME(t_pack_start = rtclock());
_lb_dist=0;
 _ub_dist=N-1;
 polyrt_loop_dist(_lb_dist, _ub_dist, nprocs, my_rank,
    ↪ &lbd_t4, &ubd_t4);
 for (t4=lbd_t4;t4<=ubd_t4;t4++) {
  for (__p=0; __p<nprocs; __p++) { receiver_list[__p]
    ↪ = 0; } sigma_C_1_0(0,t4,N,M, my_rank, nprocs,
    ↪ receiver_list); for (__p=0; __p<nprocs; __p++) {
    ↪ if (receiver_list[__p] != 0) { send_counts_C[__p]
    ↪ = pack_C_1_0(0,t4, send_buf_C[__p], send_counts_C[__p]);
    ↪ } };
 }
IF_TIME(t_pack += rtclock() - t_pack_start);
 IF_TIME(t_comm_start = rtclock());
    ↪ MPI_Alltoall(send_counts_C, 1, MPI_INT,
    ↪ recv_counts_C, 1, MPI_INT, MPI_COMM_WORLD); req_count=0;
    ↪ for (__p=0; __p<nprocs; __p++) { if (send_counts_C[__p]
    ↪ >= 1) {IF_TIME(__total_count += send_counts_C[__p]);
    ↪ MPI_Isend(send_buf_C[__p], send_counts_C[__p], MPI_DOUBLE,
    ↪ __p, 123, MPI_COMM_WORLD, &reqs[req_count++]);}}for
    ↪ (__p=0; __p<nprocs; __p++) { if(recv_counts_C[__p]
    ↪ >= 1) { MPI_Irecv(recv_buf_C+displs_C[__p], recv_counts_C[__p],
    ↪ MPI_DOUBLE, __p, 123, MPI_COMM_WORLD, &reqs[req_count++]);}}
    ↪ MPI_Waitall(req_count, reqs, stats); for (__p=0;
    ↪ __p<nprocs; __p++) { send_counts_C[__p] = 0;}for
    ↪ (__p=0; __p<nprocs; __p++) { curr_displs_C[__p] =
    ↪ displs_C[__p]; }IF_TIME(t_comm += rtclock() - t_comm_start);;
 IF_TIME(t_unpack_start = rtclock());
for (t4=0;t4<=N-1;t4++) {
  proc = pi_0(0,t4,N,M, nprocs); if ((my_rank != proc)
    ↪ && (recv_counts_C[proc] > 0)) { if
    ↪ (is_receiver_C_1_0(0,t4,N,M,my_rank,nprocs) != 0)
    ↪ { curr_displs_C[proc] = unpack_C_1_0(0,t4,recv_buf_C,curr_displs_C[proc]);
    ↪ }};
 }
IF_TIME(t_unpack += rtclock() - t_unpack_start);
 if (M >= 1) {
  IF_TIME(t_comp_start = rtclock());
_lb_dist=0;
  _ub_dist=N-1;
  polyrt_loop_dist(_lb_dist, _ub_dist, nprocs,
    ↪ my_rank, &lbd_t4, &ubd_t4);
  for (t4=lbd_t4;t4<=ubd_t4;t4++) {
   for (t5=0;t5<=t4;t5++) {
    lbv=0;
    ubv=M-1;
#pragma ivdep
#pragma vector always
    for (t6=lbv;t6<=ubv;t6++) {
     C[t4][t5] += alpha * (A[t5][t6]*B[t4][t6] +
    ↪ B[t5][t6]*A[t4][t6]);;
    }
   }
  }
IF_TIME(t_comp += rtclock() - t_comp_start);
 }
 if (M >= 1) {
  IF_TIME(t_pack_start = rtclock());
_lb_dist=0;
```

```
  _ub_dist=N-1;
  polyrt_loop_dist(_lb_dist, _ub_dist, nprocs,
    ↪ my_rank, &lbd_t4, &ubd_t4);
  for (t4=lbd_t4;t4<=ubd_t4;t4++) {
55  ;
  }
IF_TIME(t_pack += rtclock() - t_pack_start);
 }
 if (M >= 1) {
60  IF_TIME(t_comm_start = rtclock());
    ↪ MPI_Alltoall(send_counts_C, 1, MPI_INT,
    ↪ recv_counts_C, 1, MPI_INT, MPI_COMM_WORLD); req_count=0;
    ↪ for (__p=0; __p<nprocs; __p++) { if (send_counts_C[__p]
    ↪ >= 1) {IF_TIME(__total_count += send_counts_C[__p]);
    ↪ MPI_Isend(send_buf_C[__p], send_counts_C[__p], MPI_DOUBLE,
    ↪ __p, 123, MPI_COMM_WORLD, &reqs[req_count++]);}}for
    ↪ (__p=0; __p<nprocs; __p++) { if(recv_counts_C[__p]
    ↪ >= 1) { MPI_Irecv(recv_buf_C+displs_C[__p], recv_counts_C[__p],
    ↪ MPI_DOUBLE, __p, 123, MPI_COMM_WORLD, &reqs[req_count++]);}}
    ↪ MPI_Waitall(req_count, reqs, stats); for (__p=0;
    ↪ __p<nprocs; __p++) { send_counts_C[__p] = 0;}for
    ↪ (__p=0; __p<nprocs; __p++) { curr_displs_C[__p] =
    ↪ displs_C[__p]; }IF_TIME(t_comm += rtclock() - t_comm_start);;
 }
 if (M >= 1) {
  IF_TIME(t_unpack_start = rtclock());
for (t4=0;t4<=N-1;t4++) {
65   proc = pi_1(1,t4,N,M, nprocs); if ((my_rank !=
    ↪ proc) && (recv_counts_C[proc] > 0)) { };
  }
IF_TIME(t_unpack += rtclock() - t_unpack_start);
 }
}
70 /* End of CLooG code */
```

Г.5 Статистика запусков syr2k (OpenMP)

Таблица 27 — Статистика запусков syr2k в вариантах vanilla, ilp_async, pluto: затраченное время, мс

#Нитей	Стат.	vanilla	ilp_async	pluto
1	μ	26915.8054	2315.0406	2186.9346
	σ	5842.925362	19.287313	25.248569
	Min	10383.512	2292.335	2174.719
	Max	29624.081	2353.925	2260.214
2	μ	–	1757.9644	1752.9936
	σ	–	9.241413	8.360322
	Min	–	1741.756	1737.364
	Max	–	1773.438	1764.241
4	μ	–	1027.8887	1081.5861
	σ	–	17.78788	7.383384

Продолжение таблицы 27

#Нитей	Стат.	vanilla	ilp_async	pluto
	Min	–	1014.103	1075.2
	Max	–	1076.675	1099.399
6	μ	–	743.9486	772.2856
	σ	–	13.241621	5.49957
	Min	–	729.298	763.58
	Max	–	776.342	782.512
8	μ	–	586.6258	601.7906
	σ	–	10.697816	1.285665
	Min	–	577.857	599.386
	Max	–	607.069	604.582

Г.6 Статистика запусков syr2k (MPI)

Таблица 28 — Статистика запусков syr2k_mpi в вариантах vanilla, ilp_arrays_async (_one, _two, _eq): затраченное время, мс

#Проц.	Стат.	vanilla	ilp_arrays_async_one	ilp_arrays_async_two	ilp_arrays_async_eq
1	μ	12935.19074	2333.09493	–	–
	σ	9011.828882	26.105582	–	–
	Min	7809.25	2316.845	–	–
	Max	29174.952	2514.677	–	–
2	μ	–	2398.09835	2336.14513	–
	σ	–	20.660519	25.596405	–
	Min	–	2383.587	2320.087	–
	Max	–	2504.467	2520.806	–
4	μ	–	2026.22467	1855.34996	1846.60374
	σ	–	8.177486	19.74914	16.75678
	Min	–	2020.99	1843.803	1821.843
	Max	–	2087.064	1978.657	1976.004
6	μ	–	1610.05802	1556.68486	1438.16805
	σ	–	8.617088	6.566577	18.687242
	Min	–	1590.243	1547.083	1417.282
	Max	–	1646.616	1599.54	1573.573
8	μ	–	1356.04084	1299.43039	1246.66583
	σ	–	3.797394	11.389058	18.903377
	Min	–	1350.879	1280.129	1174.237
	Max	–	1387.059	1346.403	1330.936

Таблица 29 — Статистика запусков syr2k_mpi в вариантах vanilla, ilp_arrays_async (_one, _two, _eq): доля времени вычислений, %

#Проц.	Стат.	vanilla	ilp_arrays_async_one	ilp_arrays_async_two	ilp_arrays_async_eq
1	μ	100.0	99.993065	–	–
	σ	0.0	0.000548	–	–
	Min	100.0	99.991598	–	–

Продолжение таблицы 29

#Проц.	Стат.	vanilla	ilp_arrays_async_one	ilp_arrays_async_two	ilp_arrays_async_eq
	Max	100.0	99.994377	–	–
2	μ	–	49.740248	49.687667	–
	σ	–	0.005028	0.003021	–
	Min	–	49.728407	49.683464	–
	Max	–	49.756409	49.704771	–
4	μ	–	32.854273	32.192698	31.762093
	σ	–	0.031536	0.127548	0.062251
	Min	–	32.745522	31.74763	31.317541
	Max	–	33.031594	32.60238	31.878718
6	μ	–	31.153856	28.620604	27.182901
	σ	–	0.136206	0.07591	0.108832
	Min	–	30.848277	28.336022	26.417237
	Max	–	31.28637	28.753147	27.34486
8	μ	–	31.369494	27.680372	24.932393
	σ	–	0.122053	0.233452	0.174396
	Min	–	30.913895	27.005544	24.454466
	Max	–	31.480454	27.895205	25.421595

Таблица 30 — Статистика запусков syr2k_mpi в вариантах pluto (_one, _two, _eq): затраченное время, мс

#Проц.	Стат.	pluto_one	pluto_two	pluto_eq
1	μ	2313.804758	–	–
	σ	31.42913	–	–
	Min	2290.329933	–	–
	Max	2485.566139	–	–
2	μ	1851.531897	1822.543836	–
	σ	6.490944	90.954296	–
	Min	1843.274832	1747.145891	–
	Max	1902.753115	1977.396011	–
4	μ	1228.2322	1222.274711	1069.590061
	σ	2.789776	72.295048	71.021713
	Min	1224.71714	1141.489983	1038.727045
	Max	1240.93008	1301.726103	1264.117956
6	μ	927.062693	901.682427	759.030015
	σ	0.955738	56.200061	65.037779
	Min	925.884008	853.569984	728.734016
	Max	932.023048	987.931013	925.131083
8	μ	804.778762	730.138164	603.163888
	σ	0.321429	34.359158	29.241213
	Min	804.26383	698.999882	561.872959
	Max	806.112051	790.817976	724.020004

Таблица 31 — Статистика запусков syr2k_mpi в вариантах pluto (_one, _two, _eq): доля времени вычислений, %

#Проц.	Стат.	pluto_one	pluto_two	pluto_eq
1	μ	99.99741	–	–
	σ	0.000112	–	–
	Min	99.996981	–	–
	Max	99.99792	–	–
2	μ	66.245817	68.248954	–
	σ	0.120197	2.448825	–
	Min	66.073673	63.677294	–

Продолжение таблицы 31

#Проц.	Стат.	pluto_one	pluto_two	pluto_eq
	Max	67.244498	70.327267	–
4	μ	58.058389	57.071317	57.782919
	σ	0.109068	2.299058	2.862908
	Min	57.909437	54.482105	49.838977
	Max	58.439434	59.773054	63.120652
6	μ	57.296113	55.2829	53.460759
	σ	0.055734	2.46602	3.401641
	Min	57.222488	51.956612	44.56436
	Max	57.549415	57.874738	57.911356
8	μ	58.428577	53.107461	51.673267
	σ	0.032722	1.958083	1.977403
	Min	58.359503	51.003397	44.40496
	Max	58.542122	56.496685	55.533033

Таблица 32 — Разнородная нагрузка в запусках syr2k_mpi

Запуск	Затраченное время, мс			Доля затраченного времени, %		
vanilla: 1=1x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	11748.55	0.00	0.00	100.00	0.00	0.00
ilp_arrays_async_one: 1=1x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	2331.41	0.16	0.00	99.99	0.01	0.00
ilp_arrays_async_one: 2=1x2	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	2396.54	1851.29	0.01	22.75	77.25	0.00
Process 1	2396.54	10.12	547.67	76.73	0.42	22.85
ilp_arrays_async_two: 2=2x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	2334.05	1792.46	0.02	23.20	76.80	0.00
Process 1	2334.05	11.78	544.44	76.17	0.50	23.33
ilp_arrays_async_one: 4=1x4	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	2025.64	1182.11	693.27	7.42	58.36	34.22
Process 1	2025.64	1182.12	339.81	24.87	58.36	16.78
Process 2	2025.64	1182.10	0.01	41.64	58.36	0.00
Process 3	2025.64	16.65	844.99	57.46	0.82	41.71
ilp_arrays_async_two: 4=2x2	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	1853.17	1142.58	569.32	7.62	61.66	30.72
Process 1	1853.17	1142.66	286.41	22.89	61.66	15.46
Process 2	1853.17	1141.63	0.02	38.40	61.60	0.00
Process 3	1853.17	22.95	712.45	60.32	1.24	38.44
ilp_arrays_async_eq: 4=4x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	1845.08	1122.44	590.20	7.18	60.83	31.99
Process 1	1845.08	1122.45	326.03	21.49	60.83	17.67
Process 2	1845.08	1122.47	0.02	39.16	60.84	0.00
Process 3	1845.08	36.09	725.56	58.72	1.96	39.32
ilp_arrays_async_one: 6=1x6	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	1611.18	842.31	695.50	4.55	52.28	43.17
Process 1	1611.17	842.24	500.67	16.65	52.28	31.07
Process 2	1611.18	842.30	319.87	27.87	52.28	19.85
Process 3	1611.18	842.28	158.17	37.91	52.28	9.82
Process 4	1611.18	842.29	0.01	47.72	52.28	0.00
Process 5	1611.18	19.94	769.53	51.00	1.24	47.76
ilp_arrays_async_two: 6=2x3	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	1557.35	847.42	645.52	4.14	54.41	41.45
Process 1	1557.35	847.42	509.21	12.89	54.41	32.70
Process 2	1557.35	847.39	376.75	21.40	54.41	24.19
Process 3	1557.35	847.28	154.70	35.66	54.41	9.93
Process 4	1557.35	846.53	0.02	45.64	54.36	0.00
Process 5	1557.35	34.45	711.44	52.10	2.21	45.68

Продолжение таблицы 32

Запуск	Затраченное время, мс			Доля затраченного времени, %		
ilp_arrays_async_eq: 6=6x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	1436.45	839.96	537.47	4.11	58.47	37.42
Process 1	1436.47	839.99	419.88	12.29	58.48	29.23
Process 2	1436.47	839.92	301.23	20.56	58.47	20.97
Process 3	1436.47	839.86	170.98	29.63	58.47	11.90
Process 4	1436.47	839.90	0.02	41.53	58.47	0.00
Process 5	1436.47	59.45	599.13	54.15	4.14	41.71
ilp_arrays_async_one: 8=1x8	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	1355.73	661.57	649.65	3.28	48.80	47.92
Process 1	1355.73	661.46	505.42	13.93	48.79	37.28
Process 2	1355.73	661.47	384.92	22.82	48.79	28.39
Process 3	1355.73	661.72	278.36	30.66	48.81	20.53
Process 4	1355.73	661.83	180.52	37.87	48.82	13.31
Process 5	1355.73	661.83	86.76	44.78	48.82	6.40
Process 6	1355.73	661.74	0.01	51.19	48.81	0.00
Process 7	1355.73	21.36	694.70	47.18	1.58	51.24
ilp_arrays_async_two: 8=2x4	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	1299.03	686.48	575.16	2.88	52.85	44.28
Process 1	1299.03	686.73	492.66	9.21	52.86	37.93
Process 2	1299.03	686.74	398.42	16.46	52.87	30.67
Process 3	1299.03	686.59	317.42	22.71	52.85	24.44
Process 4	1299.03	685.43	177.39	33.58	52.76	13.66
Process 5	1299.03	686.83	69.97	41.74	52.87	5.39
Process 6	1299.02	686.84	0.02	47.12	52.87	0.00
Process 7	1299.03	45.63	613.50	49.26	3.51	47.23
ilp_arrays_async_eq: 8=8x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	1246.56	709.38	501.44	2.87	56.91	40.23
Process 1	1246.56	709.33	429.54	8.64	56.90	34.46
Process 2	1246.56	709.41	364.64	13.84	56.91	29.25
Process 3	1246.56	709.52	282.47	20.42	56.92	22.66
Process 4	1246.56	709.45	212.66	26.03	56.91	17.06
Process 5	1246.56	709.53	103.64	34.77	56.92	8.31
Process 6	1246.57	709.56	0.02	43.08	56.92	0.00
Process 7	1246.57	83.78	539.11	50.03	6.72	43.25
pluto_one: 1=1x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	2307.99	0.06	0.00	100.00	0.00	0.00
pluto_one: 2=1x2	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	1850.91	1249.59	0.00	32.49	67.51	0.00
Process 1	1850.91	0.07	0.00	100.00	0.00	0.00
pluto_two: 2=2x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	1833.62	1123.16	0.00	38.75	61.25	0.00
Process 1	1833.62	0.20	0.00	99.99	0.01	0.00
pluto_one: 4=1x4	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	1228.10	1058.13	0.00	13.84	86.16	0.00
Process 1	1228.10	676.11	0.00	44.95	55.05	0.00
Process 2	1228.10	321.72	0.00	73.80	26.20	0.00
Process 3	1228.10	0.07	0.00	99.99	0.01	0.00
pluto_two: 4=2x2	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	1223.04	1010.07	0.00	17.41	82.59	0.00
Process 1	1223.04	682.81	0.00	44.17	55.83	0.00
Process 2	1223.04	319.19	0.00	73.90	26.10	0.00
Process 3	1223.04	0.14	0.00	99.99	0.01	0.00
pluto_eq: 4=4x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	1063.05	843.97	0.00	20.61	79.39	0.00
Process 1	1063.05	518.19	0.00	51.25	48.75	0.00
Process 2	1063.05	357.12	0.00	66.41	33.59	0.00
Process 3	1063.05	0.16	0.00	99.99	0.01	0.00

Продолжение таблицы 32

Запуск	Затраченное время, мс			Доля затраченного времени, %		
pluto_one: 6=1x6	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	926.97	844.36	0.00	8.91	91.09	0.00
Process 1	926.97	634.28	0.00	31.57	68.43	0.00
Process 2	926.97	459.45	0.00	50.44	49.56	0.00
Process 3	926.97	296.10	0.00	68.06	31.94	0.00
Process 4	926.97	143.65	0.00	84.50	15.50	0.00
Process 5	926.97	0.07	0.00	99.99	0.01	0.00
pluto_two: 6=2x3	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	902.86	784.76	0.00	13.08	86.92	0.00
Process 1	902.86	629.76	0.00	30.25	69.75	0.00
Process 2	902.86	476.61	0.00	47.21	52.79	0.00
Process 3	902.87	293.28	0.00	67.52	32.48	0.00
Process 4	902.87	139.41	0.00	84.56	15.44	0.00
Process 5	902.87	0.24	0.00	99.97	0.03	0.00
pluto_eq: 6=6x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	751.16	691.22	0.00	7.98	92.02	0.00
Process 1	751.16	570.80	0.00	24.01	75.99	0.00
Process 2	751.16	458.53	0.00	38.96	61.04	0.00
Process 3	751.16	315.62	0.00	57.98	42.02	0.00
Process 4	751.16	0.11	0.00	99.99	0.01	0.00
Process 5	751.16	17.81	0.00	97.63	2.37	0.00
pluto_one: 8=1x8	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	804.77	752.22	0.00	6.53	93.47	0.00
Process 1	804.77	591.03	0.00	26.56	73.44	0.00
Process 2	804.77	465.48	0.00	42.16	57.84	0.00
Process 3	804.77	357.44	0.00	55.58	44.42	0.00
Process 4	804.77	260.69	0.00	67.61	32.39	0.00
Process 5	804.77	168.70	0.00	79.04	20.96	0.00
Process 6	804.77	80.50	0.00	90.00	10.00	0.00
Process 7	804.77	0.08	0.00	99.99	0.01	0.00
pluto_two: 8=2x4	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	742.45	705.14	0.00	5.02	94.98	0.00
Process 1	742.45	626.04	0.00	15.68	84.32	0.00
Process 2	742.45	538.78	0.00	27.43	72.57	0.00
Process 3	742.45	459.62	0.00	38.09	61.91	0.00
Process 4	742.45	263.24	0.00	64.54	35.46	0.00
Process 5	742.45	190.85	0.00	74.29	25.71	0.00
Process 6	742.45	80.12	0.00	89.21	10.79	0.00
Process 7	742.45	0.12	0.00	99.98	0.02	0.00
pluto_eq: 8=8x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	599.84	566.14	0.00	5.62	94.38	0.00
Process 1	599.83	421.57	0.00	29.72	70.28	0.00
Process 2	599.84	420.75	0.00	29.86	70.14	0.00
Process 3	599.83	362.86	0.00	39.51	60.49	0.00
Process 4	599.82	266.01	0.00	55.65	44.35	0.00
Process 5	599.82	204.36	0.00	65.93	34.07	0.00
Process 6	599.83	74.15	0.00	87.64	12.36	0.00
Process 7	599.83	0.09	0.00	99.99	0.01	0.00

Приложение Д

Распараллеливание программы floyd на языке C

Д.1 Описание программы

Алгоритм Флойда–Уоршалла находит кратчайшее расстояние для каждой пары вершин в графе. На вход поступает матрица весов A размера $N \times N$, где $A[i,j]$ хранит вес ребра, исходящего из вершины i в вершину j. Кратчайшие пути определяются рекурсивно:

$$p(k,i,j) = \begin{cases} A[i,j] & k = -1 \\ min(p(k-1,i,j), p(k-1,i,k) + p(k-1,k,j)) & 0 \leqslant k < N \end{cases}.$$

Листинг Д.1 Программа floyd (последовательный вариант) на языке C

```
1 void floyd_vanilla(int N, double** A) {
2 #pragma scop
3   for (int k = 0; k < N; k++)
4     for (int i = 0; i < N; i++)
5       for (int j = 0; j < N; j++)
6         A[i][j] = min(A[i][k] + A[k][j], A[i][j]); //S0
7 #pragma endscop
8 }
```

На рисунке Д.1 проиллюстрирован обобщенный граф зависимостей фрагмента функции `floyd_vanilla`.

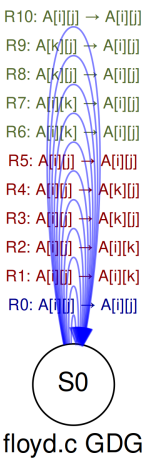


Рисунок Д.1 — Обобщенный граф зависимостей floyd_vanilla

Зависимости R1 — R5 имеют тип «чтение после записи», зависимости R6 — R10, имеют тип «запись после чтения», зависимость R0 имеет тип «запись после записи».

Д.2 Журнал трансляции ilpy

Листинг Д.2 Журнал трансляции ilpy при нахождении аффинных отображений для floyd

```
#Расписание вычислений, первый компонент

R10 (weight 990000): L=1
R8 (weight 990000): L=1
R6 (weight 990000): L=1
R5 (weight 990000): L=1
R3 (weight 990000): L=1
R1 (weight 990000): L=1
R0 (weight 990000): L=1
zero: 0, constant: 7, affine: 0, total: 7
C = 6930000.000000

#Расписание вычислений, второй компонент

R9 (weight 9900): L=1
R7 (weight 9900): L=1
R4 (weight 9900): L=1
R2 (weight 9900): L=1
zero: 0, constant: 4, affine: 0, total: 4
C = 39600.000000

#Размещение вычислений

# component 1 with linear independence ENABLED and FCO property DISABLED
R10 (weight 990000): L=0
R9 (weight 9900): L=0
R8 (weight 990000): L=0
R7 (weight 9900): L=1
R6 (weight 990000): L=N
R5 (weight 990000): L=0
R4 (weight 9900): L=0
R3 (weight 990000): L=0
R2 (weight 9900): L=1
R1 (weight 990000): L=N
R0 (weight 990000): L=0
zero: 7, constant: 2, affine: 2, total: 11
C = 198019800.000000
```

Листинг Д.3 Журнал трансляции ilpy при нахождении размещения вычислений и данных для floyd

```
# component 1 with linear independence ENABLED
    and FCO property DISABLED
S0, A3 (READ A[i][j], weight 1000000): L=0
S0, A2 (READ A[k][j], weight 1000000): L=0
S0, A1 (READ A[i][k], weight 1000000): L=N
S0, A0 (WRITE A[i][j], weight 1000000): L=0
zero: 3, constant: 0, affine: 1, total: 4
C = 100000000.000000
```

Д.3 Результат работы ilpy

Листинг Д.4 floyd (параллельный вариант ilpy без директив OpenMP, синхронный параллелизм) на языке C

```
if (N >= 1) {
 for (t0=0;t0<=N-1;t0++) {
  for (t1=0;t1<=2*N-2;t1++) {
   for (ilpp=max(0,t1-N+1);ilpp<=min(t1,N-1);ilpp++) {
    A[t1-ilpp][ilpp] = A[t1-ilpp][t0] + A[t0][ilpp] + A[t1-ilpp][ilpp];
   }
  }
 }
}
```

Листинг Д.5 floyd_ilp_sync (параллельный вариант ilpy с директивами OpenMP, синхронный параллелизм) на языке C

```
void floyd_ilp_sync(int N, double** A) {
  #pragma omp parallel
  {
    if (N >= 1) {
      for (int t0=0;t0<=N-1;t0++) {
        for (int t1=0;t1<=2*N-2;t1++) {
          #pragma omp for
          for (int ilpp=max(0,t1-N+1);ilpp<=min(t1,N-1);ilpp++) {
            A[t1-ilpp][ilpp] = min(A[t1-ilpp][t0] + A[t0][ilpp], A[t1-ilpp][ilpp]);
          }
        }
      }
    }
  }
}
```

Листинг Д.6 floyd (параллельный вариант ilpy без конструкций MPI, синхронный параллелизм) на языке C

```
if (N >= 1) {
 for (t0=0;t0<=N-1;t0++) {
  for (t1=0;t1<=2*N-2;t1++) {
   for (ilpp=max(0,t1-N+1);ilpp<=min(t1,N-1);ilpp++) {
    A[t1-ilpp][ilpp] = A[t1-ilpp][t0] + A[t0][ilpp] + A[t1-ilpp][ilpp];
   }
  }
 }
}
```

Листинг Д.7 floyd_ilp_arrays (параллельный вариант ilpy с конструкциями MPI, синхронный параллелизм) на языке C

```
// arrays placement

#define T_base 1000000

#define eta_A(i0,i1) (i1)
#define T_A (1 * T_base)

int A_chunk_size;
int A_actual_chunk_size;

// mpi routine

#ifdef COL_MAJOR
#define _A(row, col) A[col][row]
#define eta_A_t0(i0) eta_A(i0, t0)
#else
double* buf_A;
#endif

void floyd_ilp_arrays(int N, double** A) {
  if (N >= 1) {
    for (int t0=0;t0<=N-1;t0++) {
      for (int t1=0;t1<=2*N-2;t1++) {
        // A[t1-ilpp][t0] in A[t1-ilpp][ilpp] = min(A[t1-ilpp][t0] + A[t0][ilpp], A[t1-ilpp][ilpp])
#ifdef COL_MAJOR
        line_Q_read_vp_rev(-1, t1, max(0,t1-N+1), min(t1,N-1), A[t0], eta_A_t0, T_A + t0);
#else
        col_Q_read_vp_rev(-1, t1, max(0,t1-N+1), min(t1,N-1), t0, buf_A, A, eta_A, T_A + t0);
#endif
        //
        for (int ilpp=max(lR,max(0,t1-N+1));ilpp<=min(uR,min(t1,N-1));ilpp++) {
#ifdef COL_MAJOR
          _A(t1-ilpp, ilpp) = min(_A(t1-ilpp, t0) + _A(t0, ilpp), _A(t1-ilpp, ilpp));
#else
          A[t1-ilpp][ilpp] = min(A[t1-ilpp][t0] + A[t0][ilpp], A[t1-ilpp][ilpp]);
#endif
        }
        MPI_Barrier_time(MPI_COMM_WORLD);
      }
    }
#ifdef COL_MAJOR
    collect_matrix_rows_double(A, N, N, A_chunk_size);
```

```
#else
    collect_matrix_cols_double(A, N, N, A_chunk_size, false);
#endif
  }
}
```

Д.4 Преобразования pluto

Листинг Д.8 Преобразования pluto в формате cloog для floyd_pluto

```
# Number of scattering functions
1

# T(S1)
3 9
0 1 0 0 -1 0 0 0 0
0 0 1 0 0 -1 -1 0 0
0 0 0 1 0 0 -1 0 0
```

Листинг Д.9 floyd (параллельный вариант pluto с директивами OpenMP, синхронный параллелизм) на языке C

```
 int t1, t2, t3;
 int lb, ub, lbd, ubd, lb2, ub2;
 register int lbv, ubv;
/* Start of CLooG code */
if (N >= 1) {
 for (t1=0;t1<=N-1;t1++) {
  for (t2=0;t2<=2*N-2;t2++) {
   {/* extra braces to avoid redefination of lbp*/
int lbp=max(0,t2-N+1);
   int ubp=min(t2,N-1);
#pragma omp parallel for private(lbv,ubv)
   for (t3=lbp;t3<=ubp;t3++) {
    A[(t2-t3)][t3] = A[(t2-t3)][t1] + A[t1][t3] + A[(t2-t3)][t3];;
   }
}/*end of omp parallel loop */
  }
 }
}
/* End of CLooG code */
```

Листинг Д.10 floyd_pluto (обработанный вариант pluto с директивами OpenMP, синхронный параллелизм) на языке C

```
void floyd_pluto(int N, double** A) {
  #pragma omp parallel
  {
    /* Start of CLooG code */
    if (N >= 1) {
```

```
      for (int t1=0;t1<=N-1;t1++) {
        for (int t2=0;t2<=2*N-2;t2++) {
          #pragma omp for
          for (int t3=max(0,t2-N+1);t3<=min(t2,N-1);t3++) {
            A[(t2-t3)][t3] = min(A[(t2-t3)][t1] + A[t1][t3], A[(t2-t3)][t3]);
          }
        }
      }
    }
    /* End of CLooG code */
  }
}
```

Листинг Д.11 Преобразования pluto в формате cloog для floyd_pluto_mpi

```
# Number of scattering functions
4

# T(S1)
4 11
0 1 0 0 0 -1 0 0 0 0 0
0 0 1 0 0 0 -1 -1 0 0 0
0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 1 0 0 -1 0 0 0
```

Листинг Д.12 floyd_pluto_mpi (параллельный вариант pluto с конструкциями MPI, синхронный параллелизм) на языке C

```
/* Start of CLooG code */
if (N >= 1) {
 for (t1=0;t1<=N-1;t1++) {
  for (t2=0;t2<=2*N-2;t2++) {
    IF_TIME(t_comp_start = rtclock());
_lb_dist=max(0,t2-N+1);
   _ub_dist=min(t2,N-1);
   polyrt_loop_dist(_lb_dist, _ub_dist, nprocs,
    ↪ my_rank, &lbd_t4, &ubd_t4);
   for (t4=lbd_t4;t4<=ubd_t4;t4++) {
    A[(t2-t4)][t4] = min(A[(t2-t4)][t1] + A[t1][t4],
    ↪ A[(t2-t4)][t4]);;
   }
IF_TIME(t_comp += rtclock() - t_comp_start);
   IF_TIME(t_pack_start = rtclock());
_lb_dist=max(0,t2-N+1);
   _ub_dist=min(t2,N-1);
   polyrt_loop_dist(_lb_dist, _ub_dist, nprocs,
    ↪ my_rank, &lbd_t4, &ubd_t4);
   for (t4=lbd_t4;t4<=ubd_t4;t4++) {
    for (__p=0; __p<nprocs; __p++) {
    ↪ receiver_list[__p] = 0; }
    ↪ sigma_A_1_0(t1,t2,t4,N, my_rank, nprocs, receiver_list);
    ↪ for (__p=0; __p<nprocs; __p++) { if (receiver_list[__p]
    ↪ != 0) { send_counts_A[__p] = pack_A_1_0(t1,t2,t4,
    ↪ send_buf_A[__p], send_counts_A[__p]); } };
   }
IF_TIME(t_pack += rtclock() - t_pack_start);
   IF_TIME(t_comm_start = rtclock());
    ↪ MPI_Alltoall(send_counts_A, 1, MPI_INT,
    ↪ recv_counts_A, 1, MPI_INT, MPI_COMM_WORLD); req_count=0;
```

```
    ↪ for (__p=0; __p<nprocs; __p++) { if (send_counts_A[__p]
    ↪ >= 1) {IF_TIME(__total_count += send_counts_A[__p]);
    ↪ MPI_Isend(send_buf_A[__p], send_counts_A[__p], MPI_DOUBLE,
    ↪ __p, 123, MPI_COMM_WORLD, &reqs[req_count++]);}}for
    ↪ (__p=0; __p<nprocs; __p++) { if(recv_counts_A[__p]
    ↪ >= 1) { MPI_Irecv(recv_buf_A+displs_A[__p], recv_counts_A[__p],
    ↪ MPI_DOUBLE, __p, 123, MPI_COMM_WORLD, &reqs[req_count++]);}}
    ↪ MPI_Waitall(req_count, reqs, stats); for (__p=0;
    ↪ __p<nprocs; __p++) { send_counts_A[__p] = 0;}for
    ↪ (__p=0; __p<nprocs; __p++) { curr_displs_A[__p] =
    ↪ displs_A[__p]; }IF_TIME(t_comm += rtclock() - t_comm_start);;
   IF_TIME(t_unpack_start = rtclock());
for (t4=max(0,t2-N+1);t4<=min(t2,N-1);t4++) {
    proc = pi_0(t1,t2,t4,N, nprocs); if ((my_rank !=
    ↪ proc) && (recv_counts_A[proc] > 0)) { if
    ↪ (is_receiver_A_1_0(t1,t2,t4,N,my_rank,nprocs) !=
    ↪ 0) { curr_displs_A[proc] = unpack_A_1_0(t1,t2,t4,recv_buf_A,curr_displs_A[proc]);
    ↪ }};
   }
IF_TIME(t_unpack += rtclock() - t_unpack_start);
  }
 }
}
/* End of CLooG code */
```

Д.5 Статистика запусков floyd (OpenMP)

Таблица 33 — Статистика запусков floyd в вариантах vanilla, ilp_sync, pluto: затраченное время, мс

#Нитей	Стат.	vanilla	ilp_sync	pluto
1	μ	937.4961	11762.3626	11784.8554
	σ	61.895931	14.479941	12.579
	Min	914.722	11753.822	11774.055
	Max	1123.041	11805.563	11810.663
2	μ	–	5607.9562	5589.4339
	σ	–	2.146256	1.227345
	Min	–	5605.604	5587.977
	Max	–	5613.22	5591.684
4	μ	–	1843.9441	1828.3608
	σ	–	4.730029	20.249876
	Min	–	1841.152	1819.944
	Max	–	1857.634	1889.062
6	μ	–	1563.8184	1579.1969
	σ	–	1.648655	0.98854
	Min	–	1561.632	1578.083
	Max	–	1568.016	1581.192
8	μ	–	1467.7801	1456.7061
	σ	–	1.886599	0.581743
	Min	–	1465.079	1455.228
	Max	–	1472.868	1457.585

Д.6 Статистика запусков floyd (MPI)

Таблица 34 — Статистика запусков floyd_mpi в вариантах vanilla, ilp_arrays (_one, _two, _eq): затраченное время, мс

#Проц.	Стат.	vanilla	ilp_arrays_one	ilp_arrays_two	ilp_arrays_eq
1	μ	944.10414	10626.78229	–	–
	σ	22.891776	18.116881	–	–
	Min	914.001	10613.732	–	–
	Max	1107.673	10744.762	–	–
2	μ	–	13963.50008	25606.29544	–
	σ	–	12.620597	15.563043	–
	Min	–	13952.555	25577.673	–
	Max	–	14037.026	25707.745	–
4	μ	–	11008.22293	19709.4509	28654.47485
	σ	–	9.507425	81.791555	131.079364
	Min	–	10989.801	19617.582	28329.86
	Max	–	11034.555	19983.259	28940.383
6	μ	–	10412.9358	19190.09679	30397.1451
	σ	–	9.984411	547.682095	61.768463
	Min	–	10385.308	18302.5	30271.618
	Max	–	10427.692	19718.624	30528.397
8	μ	–	10518.49792	18704.23154	29523.47778
	σ	–	11.050102	35.806868	45.917065
	Min	–	10503.851	18625.173	29415.711
	Max	–	10534.875	18837.454	29692.774

Таблица 35 — Статистика запусков floyd_mpi в вариантах vanilla, ilp_arrays (_one, _two, _eq): доля времени вычислений, %

#Проц.	Стат.	vanilla	ilp_arrays_one	ilp_arrays_two	ilp_arrays_eq
1	μ	100.0	98.913962	–	–
	σ	0.0	0.003237	–	–
	Min	100.0	98.905938	–	–
	Max	100.0	98.921611	–	–
2	μ	–	31.696172	17.205766	–
	σ	–	0.007433	0.005993	–
	Min	–	31.634884	17.194287	–
	Max	–	31.710544	17.237128	–
4	μ	–	11.269997	6.557966	4.953118
	σ	–	0.008018	0.026816	0.016804
	Min	–	11.255978	6.468477	4.90791
	Max	–	11.306685	6.589878	4.989305
6	μ	–	6.339021	3.676709	2.213287
	σ	–	0.005729	0.025851	0.00533
	Min	–	6.32438	3.594747	2.199991
	Max	–	6.353184	3.7143	2.223374
8	μ	–	4.5128	2.687816	1.613542
	σ	–	0.004047	0.018857	0.00236
	Min	–	4.499875	2.602138	1.607201
	Max	–	4.523757	2.704008	1.619687

Таблица 36 — Статистика запусков floyd_mpi в вариантах pluto (_one, _two, _eq): затраченное время, мс

#Проц.	Стат.	pluto_one	pluto_two	pluto_eq
1	μ	57370.220673	–	–
	σ	1661.843519	–	–
	Min	56454.336882	–	–
	Max	60722.00489	–	–
2	μ	46156.116474	52459.205465	–
	σ	1406.640569	85.500415	–
	Min	45331.349134	52139.014006	–
	Max	48813.288927	52721.871853	–
4	μ	42403.805797	45398.405902	47320.415285
	σ	1020.244868	91.294368	247.351809
	Min	39723.310947	45060.263157	46885.161161
	Max	42976.541996	45633.665085	47858.487129
6	μ	40652.823896	49495.434318	40549.622645
	σ	618.702773	1126.488456	52.448407
	Min	37924.91889	44283.941984	40425.513983
	Max	41163.172007	50432.288885	40667.323828
8	μ	39705.137129	47762.399836	41348.458846
	σ	231.470157	975.11923	50.716777
	Min	37532.792091	45731.108904	41252.146959
	Max	39930.077076	48636.87396	41475.924015

Таблица 37 — Статистика запусков floyd_mpi в вариантах pluto (_one, _two, _eq): доля времени вычислений, %

#Проц.	Стат.	pluto_one	pluto_two	pluto_eq
1	μ	18.933387	–	–
	σ	0.004605	–	–
	Min	18.921442	–	–
	Max	18.943518	–	–
2	μ	9.478109	8.842021	–
	σ	0.015243	0.007868	–
	Min	9.461975	8.822326	–
	Max	9.514565	8.857152	–
4	μ	2.966256	2.994494	2.849661
	σ	0.044934	0.030765	0.006848
	Min	2.939225	2.973451	2.830839
	Max	3.087163	3.08349	2.864989
6	μ	1.925091	1.773878	1.882101
	σ	0.022997	0.024792	0.002819
	Min	1.910806	1.755936	1.875112
	Max	2.028864	1.895242	1.887813
8	μ	1.63226	1.420983	1.466528
	σ	0.007678	0.020441	0.00191
	Min	1.624336	1.403328	1.46195
	Max	1.702492	1.463237	1.471348

Таблица 38 — Разнородная нагрузка в запусках floyd_mpi

Запуск	Затраченное время, мс			Доля затраченного времени, %		
vanilla: 1=1x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	941.46	0.00	0.00	100.00	0.00	0.00
ilp_arrays_one: 1=1x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	10624.77	55.11	60.49	98.91	0.52	0.57

Продолжение таблицы 38

Запуск	Затраченное время, мс			Доля затраченного времени, %		
ilp_arrays_one: 2=1x2	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	13962.27	6560.68	2946.66	31.91	46.99	21.10
Process 1	13962.27	6559.57	3005.71	31.49	46.98	21.53
ilp_arrays_two: 2=2x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	25605.05	16304.50	4884.47	17.25	63.68	19.08
Process 1	25604.99	16395.29	4816.78	17.16	64.03	18.81
ilp_arrays_one: 4=1x4	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	11008.40	7572.15	2242.62	10.84	68.79	20.37
Process 1	11008.40	7577.42	2142.00	11.71	68.83	19.46
Process 2	11008.40	7574.36	2152.49	11.64	68.81	19.55
Process 3	11008.40	7571.21	2238.91	10.89	68.78	20.34
ilp_arrays_two: 4=2x2	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	19702.22	14274.46	4095.73	6.76	72.45	20.79
Process 1	19702.22	14290.82	4145.13	6.43	72.53	21.04
Process 2	19702.23	14196.26	4092.91	7.17	72.05	20.77
Process 3	19702.23	14178.72	4360.56	5.90	71.97	22.13
ilp_arrays_eq: 4=4x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	28656.50	20721.16	6569.29	4.77	72.31	22.92
Process 1	28656.50	20488.37	6703.15	5.11	71.50	23.39
Process 2	28656.50	20525.15	6670.95	5.10	71.62	23.28
Process 3	28656.50	19950.46	7319.36	4.84	69.62	25.54
ilp_arrays_one: 6=1x6	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	10413.25	8022.21	1751.24	6.14	77.04	16.82
Process 1	10413.25	8017.69	1728.50	6.41	77.00	16.60
Process 2	10413.25	8015.42	1710.74	6.60	76.97	16.43
Process 3	10413.25	8020.28	1702.49	6.63	77.02	16.35
Process 4	10413.25	8020.55	1743.16	6.24	77.02	16.74
Process 5	10413.25	8024.06	1759.60	6.05	77.06	16.90
ilp_arrays_two: 6=2x3	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	19393.50	14888.55	3797.22	3.65	76.77	19.58
Process 1	19393.50	14874.50	3850.18	3.45	76.70	19.85
Process 2	19393.50	14907.20	3766.28	3.71	76.87	19.42
Process 3	19393.47	14433.45	4203.73	3.90	74.42	21.68
Process 4	19393.47	14434.40	4280.34	3.50	74.43	22.07
Process 5	19393.47	14437.44	4293.96	3.41	74.44	22.14
ilp_arrays_eq: 6=6x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	30399.35	21862.50	7889.32	2.13	71.92	25.95
Process 1	30399.32	21127.20	8589.12	2.25	69.50	28.25
Process 2	30399.34	21744.05	7950.45	2.32	71.53	26.15
Process 3	30399.31	20544.08	9164.31	2.27	67.58	30.15
Process 4	30399.32	21980.93	7753.02	2.19	72.31	25.50
Process 5	30399.33	21262.86	8496.04	2.11	69.95	27.95
ilp_arrays_one: 8=1x8	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	10515.09	8580.09	1479.24	4.33	81.60	14.07
Process 1	10515.09	8563.40	1482.81	4.46	81.44	14.10
Process 2	10515.09	8557.12	1484.72	4.50	81.38	14.12
Process 3	10515.09	8554.62	1468.99	4.67	81.36	13.97
Process 4	10515.09	8554.79	1470.45	4.66	81.36	13.98
Process 5	10515.09	8563.87	1475.16	4.53	81.44	14.03
Process 6	10515.09	8570.35	1471.81	4.50	81.51	14.00
Process 7	10515.09	8575.63	1475.26	4.41	81.56	14.03
ilp_arrays_two: 8=2x4	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	18701.26	14883.33	3323.24	2.65	79.58	17.77
Process 1	18701.26	14884.30	3346.07	2.52	79.59	17.89
Process 2	18701.26	14896.29	3253.19	2.95	79.65	17.40
Process 3	18701.26	14889.58	3318.66	2.64	79.62	17.75
Process 4	18701.26	14149.61	4016.13	2.86	75.66	21.48

Продолжение таблицы 38

Запуск	Затраченное время, мс			Доля затраченного времени, %		
Process 5	18701.26	14143.04	4080.57	2.55	75.63	21.82
Process 6	18701.26	14158.95	4002.42	2.89	75.71	21.40
Process 7	18701.26	14161.23	4074.81	2.49	75.72	21.79
ilp_arrays_eq: 8=8x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	29521.52	22824.49	6235.51	1.56	77.31	21.12
Process 1	29521.79	22076.12	6970.21	1.61	74.78	23.61
Process 2	29521.77	21766.01	7284.13	1.60	73.73	24.67
Process 3	29521.78	20675.97	8350.34	1.68	70.04	28.29
Process 4	29521.77	21832.95	7196.92	1.67	73.96	24.38
Process 5	29521.78	20731.86	8308.75	1.63	70.23	28.14
Process 6	29521.76	20449.60	8604.56	1.58	69.27	29.15
Process 7	29521.78	19579.80	9476.41	1.58	66.32	32.10
pluto_one: 1=1x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	56608.33	45889.05	0.00	18.94	81.06	0.00
pluto_one: 2=1x2	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	45444.94	41168.05	0.00	9.41	90.59	0.00
Process 1	45444.94	41112.50	0.00	9.53	90.47	0.00
pluto_two: 2=2x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	52460.76	47842.13	0.00	8.80	91.20	0.00
Process 1	52460.81	47795.85	0.00	8.89	91.11	0.00
pluto_one: 4=1x4	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	42644.37	41441.15	0.00	2.82	97.18	0.00
Process 1	42644.37	41379.57	0.00	2.97	97.03	0.00
Process 2	42644.37	41270.32	0.00	3.22	96.78	0.00
Process 3	42644.37	41447.32	0.00	2.81	97.19	0.00
pluto_two: 4=2x2	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	45403.08	43998.84	0.00	3.09	96.91	0.00
Process 1	45403.08	43986.78	0.00	3.12	96.88	0.00
Process 2	45403.13	44181.26	0.00	2.69	97.31	0.00
Process 3	45403.13	44029.98	0.00	3.02	96.98	0.00
pluto_eq: 4=4x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	47319.60	46019.84	0.00	2.75	97.25	0.00
Process 1	47319.59	45934.09	0.00	2.93	97.07	0.00
Process 2	47319.60	45967.96	0.00	2.86	97.14	0.00
Process 3	47319.60	45969.55	0.00	2.85	97.15	0.00
pluto_one: 6=1x6	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	40792.86	40037.32	0.00	1.85	98.15	0.00
Process 1	40792.86	40025.03	0.00	1.88	98.12	0.00
Process 2	40792.86	39900.46	0.00	2.19	97.81	0.00
Process 3	40792.86	40037.60	0.00	1.85	98.15	0.00
Process 4	40792.86	40030.44	0.00	1.87	98.13	0.00
Process 5	40792.86	40032.66	0.00	1.86	98.14	0.00
pluto_two: 6=2x3	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	49594.03	48753.84	0.00	1.69	98.31	0.00
Process 1	49594.03	48773.38	0.00	1.65	98.35	0.00
Process 2	49594.03	48626.34	0.00	1.95	98.05	0.00
Process 3	49594.07	48760.95	0.00	1.68	98.32	0.00
Process 4	49594.07	48768.42	0.00	1.66	98.34	0.00
Process 5	49594.07	48615.70	0.00	1.97	98.03	0.00
pluto_eq: 6=6x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	40550.28	39785.82	0.00	1.89	98.11	0.00
Process 1	40550.25	39792.04	0.00	1.87	98.13	0.00
Process 2	40550.23	39765.68	0.00	1.93	98.07	0.00
Process 3	40550.21	39791.87	0.00	1.87	98.13	0.00
Process 4	40550.11	39795.82	0.00	1.86	98.14	0.00
Process 5	40550.22	39793.80	0.00	1.87	98.13	0.00
pluto_one: 8=1x8	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация

Продолжение таблицы 38

Запуск	Затраченное время, мс			Доля затраченного времени, %		
Process 0	39727.34	39078.15	0.00	1.63	98.37	0.00
Process 1	39727.34	39093.87	0.00	1.59	98.41	0.00
Process 2	39727.34	39002.23	0.00	1.83	98.17	0.00
Process 3	39727.34	39071.61	0.00	1.65	98.35	0.00
Process 4	39727.34	39071.71	0.00	1.65	98.35	0.00
Process 5	39727.34	39099.44	0.00	1.58	98.42	0.00
Process 6	39727.34	39100.38	0.00	1.58	98.42	0.00
Process 7	39727.34	39104.65	0.00	1.57	98.43	0.00
pluto_two: 8=2x4	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	47826.16	47138.64	0.00	1.44	98.56	0.00
Process 1	47826.16	47179.20	0.00	1.35	98.65	0.00
Process 2	47826.16	47107.95	0.00	1.50	98.50	0.00
Process 3	47826.16	47176.75	0.00	1.36	98.64	0.00
Process 4	47826.15	47159.83	0.00	1.39	98.61	0.00
Process 5	47826.15	47174.12	0.00	1.36	98.64	0.00
Process 6	47826.15	47070.32	0.00	1.58	98.42	0.00
Process 7	47826.15	47175.82	0.00	1.36	98.64	0.00
pluto_eq: 8=8x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	41348.36	40713.86	0.00	1.53	98.47	0.00
Process 1	41348.39	40748.67	0.00	1.45	98.55	0.00
Process 2	41348.37	40748.28	0.00	1.45	98.55	0.00
Process 3	41348.38	40751.55	0.00	1.44	98.56	0.00
Process 4	41348.38	40747.18	0.00	1.45	98.55	0.00
Process 5	41348.37	40751.89	0.00	1.44	98.56	0.00
Process 6	41348.36	40712.47	0.00	1.54	98.46	0.00
Process 7	41348.37	40758.51	0.00	1.43	98.57	0.00

Приложение Е

Распараллеливание программы gramschmidt на языке C

Е.1 Описание программы

Процедура gramschmidt предполагает QR-разложение матрицы с применением модифицированного процесса Грама–Шмидта [124]. Входные данные: матрица A размера $M \times N$ ранга N ($M > N$). Выходные данные: ортогональная матрица Q размера $M \times N$ и верхняя треугольная матрица R размера $N \times N$ такие, что $A = QR$.

Листинг Е.1 Программа gramschmidt (последовательный вариант) на языке C

```
1  void gramschmidt_vanilla(int M, int N, double** A, double** Q, double** R, double* nrm) {
2  #pragma scop
3    for (int k = 0; k < N; k++) {
4      *nrm = 0.0; //S0
5      for (int i = 0; i < M; i++)
6        *nrm += A[i][k] * A[i][k]; //S1
7      R[k][k] = sqrt(*nrm); //S2
8      for (int i = 0; i < M; i++)
9        Q[i][k] = A[i][k] / R[k][k]; //S3
10     for (int j = k + 1; j < N; j++) {
11       R[k][j] = 0.0; //S4
12       for (int i = 0; i < M; i++)
13         R[k][j] += Q[i][k] * A[i][j]; //S5
14       for (int i = 0; i < M; i++)
15         A[i][j] = A[i][j] - Q[i][k] * R[k][j]; //S6
16     }
17   }
18 #pragma endscop
19 }
```

На рисунке Е.1 проиллюстрирован обобщенный граф зависимостей фрагмента функции `gramschmidt_vanilla`. Зависимости R1, R5, R9, R10, R13, R15 — R21, R23, R25, R27, R30, R32 имеют тип «чтение после записи», зависимости R3, R6, R7, R8, R14, R24, R28, R29, R33 имеют тип «запись после чтения», зависимости R0, R2, R4, R11, R12, R22, R26, R31 имеют тип «запись после записи».

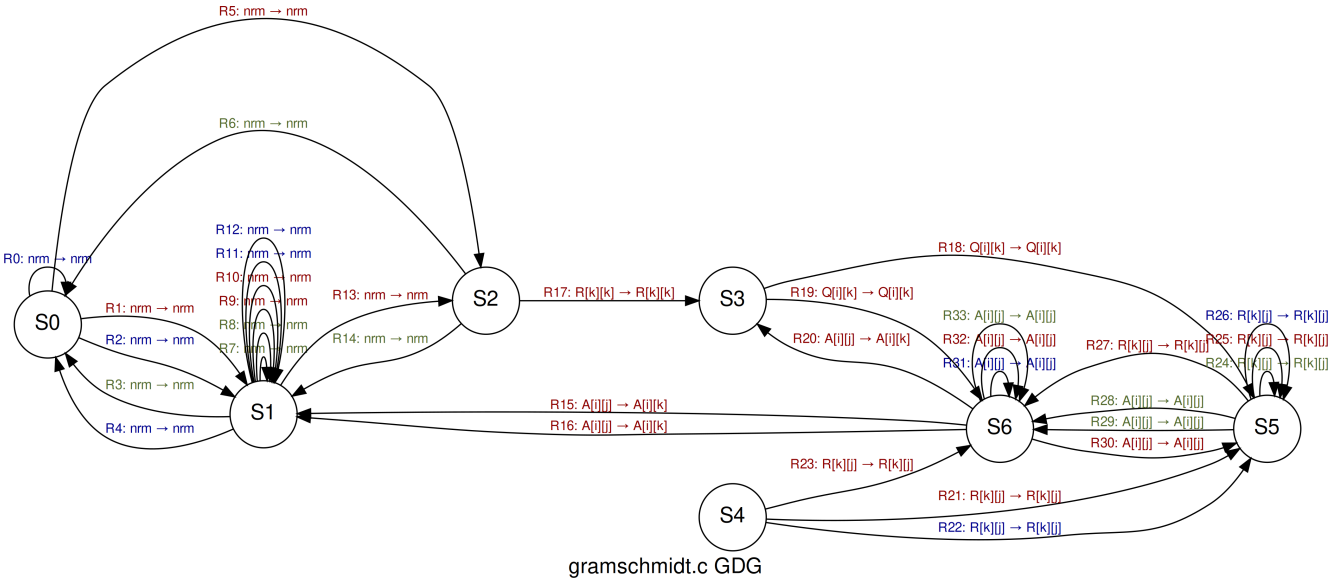


Рисунок Е.1 — Обобщенный граф зависимостей gramschmidt_vanilla

Е.2 Журнал трансляции ilpy

Листинг Е.2 Журнал трансляции ilpy при нахождении аффинных отображений для gramschmidt

```
#Расписание вычислений, первый компонент

R33 (weight 489951): L=5
R32 (weight 489951): L=5
R31 (weight 489951): L=5
R30 (weight 489951): L=4
R29 (weight 499950): L=1
R28 (weight 489951): L=6
R27 (weight 50494950): L=1
R23 (weight 499950): L=2
R22 (weight 499950): L=1
R21 (weight 499950): L=1
R20 (weight 9999): L=3
R19 (weight 499950): L=2
R18 (weight 499950): L=1
R17 (weight 10100): L=1
R16 (weight 9999): L=1
R15 (weight 9999): L=1
R14 (weight 9999): L=4
R13 (weight 10100): L=1
R11 (weight 1009899): L=5
R9 (weight 1009899): L=5
R7 (weight 1009899): L=5
R6 (weight 99): L=3
R5 (weight 100): L=2
R4 (weight 9999): L=4
R3 (weight 9999): L=4
R2 (weight 10100): L=1
R1 (weight 10100): L=1
R0 (weight 99): L=5
```

```
zero: 0, constant: 28, affine: 0, total: 28
C = 82103185.000000
```

```
#Расписание вычислений, второй компонент
```

```
R26 (weight 495000): L=1
R25 (weight 495000): L=1
R24 (weight 495000): L=1
R12 (weight 10000): L=1
R10 (weight 10000): L=1
R8 (weight 10000): L=1
zero: 0, constant: 6, affine: 0, total: 6
C = 1515000.000000
```

```
#Размещение вычислений
```

```
# component 1 with linear independence ENABLED and FCO property DISABLED
[Warning] mip_status == GLP_FEAS
R33 (weight 489951): L=0
R32 (weight 489951): L=0
R31 (weight 489951): L=0
R30 (weight 489951): L=0
R29 (weight 499950): L=0
R28 (weight 489951): L=0
R27 (weight 50494950): L=M
R26 (weight 495000): L=1
R25 (weight 495000): L=1
R24 (weight 495000): L=1
R23 (weight 499950): L=M
R22 (weight 499950): L=M
R21 (weight 499950): L=M
R20 (weight 9999): L=0
R19 (weight 499950): L=N
R18 (weight 499950): L=N
R17 (weight 10100): L=M
R16 (weight 9999): L=M
R15 (weight 9999): L=M
R14 (weight 9999): L=1
R13 (weight 10100): L=0
R12 (weight 10000): L=0
R11 (weight 1009899): L=1
R10 (weight 10000): L=0
R9 (weight 1009899): L=1
R8 (weight 10000): L=0
R7 (weight 1009899): L=1
R6 (weight 99): L=1
R5 (weight 100): L=0
R4 (weight 9999): L=1
R3 (weight 9999): L=1
R2 (weight 10100): L=0
R1 (weight 10100): L=0
R0 (weight 99): L=1
zero: 14, constant: 11, affine: 9, total: 34
C = 1064082294.000000
```

Листинг E.3 Журнал трансляции ilpy при нахождении размещения вычислений и данных для gramschmidt

```
# component 1 with linear independence ENABLED and FCO property DISABLED
S0, A0 (WRITE nrm, weight 100): L=N
S1, A3 (READ A[i][k], weight 10100): L=0
S1, A2 (READ A[i][k], weight 10100): L=0
S1, A1 (WRITE nrm, weight 10100): L=N
S1, A0 (READ nrm, weight 10100): L=N
S2, A1 (READ nrm, weight 100): L=N
S2, A0 (WRITE R[k][k], weight 100): L=0
S3, A2 (READ R[k][k], weight 10100): L=M
S3, A1 (READ A[i][k], weight 10100): L=M
S3, A0 (WRITE Q[i][k], weight 10100): L=M
S4, A0 (WRITE R[k][j], weight 4950): L=0
S5, A3 (READ A[i][j], weight 499950): L=0
S5, A2 (READ Q[i][k], weight 499950): L=N
S5, A1 (WRITE R[k][j], weight 499950): L=0
S5, A0 (READ R[k][j], weight 499950): L=0
S6, A3 (READ R[k][j], weight 499950): L=0
S6, A2 (READ Q[i][k], weight 499950): L=N
S6, A1 (READ A[i][j], weight 499950): L=0
S6, A0 (WRITE A[i][j], weight 499950): L=0
zero: 10, constant: 0, affine: 9, total: 19
C = 105090300.000000
```

E.3 Результат работы ilpy

Листинг E.4 gramschmidt (параллельный вариант ilpy без директив OpenMP, синхронный параллелизм) на языке C

```
if (N >= 1) {
 if (M >= 1) {
  for (ilpp=N-1;ilpp<=N-1;ilpp++) {
   nrm = 0.0;
  }
 }
 if (M <= 0) {
  for (ilpp=N-1;ilpp<=N-1;ilpp++) {
   nrm = 0.0;
  }
 }
 if ((M >= 1) && (N == 1)) {
  for (t1=0;t1<=M-1;t1++) {
   for (ilpp=0;ilpp<=0;ilpp++) {
    nrm += A[t1][0] * A[t1][0];
   }
  }
 }
 if ((M >= 1) && (N == 1)) {
  for (ilpp=0;ilpp<=0;ilpp++) {
   R[0][0] = nrm;
  }
 }
 if ((M >= 1) && (N >= 2)) {
  for (t1=0;t1<=M-1;t1++) {
   for (ilpp=N-1;ilpp<=N-1;ilpp++) {
```

```
    nrm += A[t1][0] * A[t1][0];
   }
  }
}
if ((M <= 0) && (N >= 2)) {
 for (ilpp=N-1;ilpp<=N-1;ilpp++) {
  R[0][0] = nrm;
 }
}
if ((M >= 1) && (N >= 2)) {
 for (ilpp=N-1;ilpp<=N-1;ilpp++) {
  R[0][0] = nrm;
 }
}
if (M <= 0) {
 for (t0=3;t0<=5*N-7;t0++) {
  for (ilpp=(t0+5*N-7)/5;ilpp<=(t0+5*N-7)/5;ilpp++) {
   if ((t0+3)%5 == 0) {
    R[(t0-2)/5][(t0-2)/5] = nrm;
   }
  }
  for (ilpp=(t0+5*N-5)/5;ilpp<=(t0+5*N-5)/5;ilpp++) {
   if (t0%5 == 0) {
    nrm = 0.0;
   }
  }
  for (ilpp=ceild(t0+5*N-8,5);ilpp<=2*N-3;ilpp++) {
   if ((t0+2)%5 == 0) {
    R[(t0-3)/5][ilpp-N+2] = 0.0;
   }
  }
 }
}
if ((M <= 0) && (N >= 2)) {
 for (ilpp=2*N-2;ilpp<=2*N-2;ilpp++) {
  nrm = 0.0;
 }
}
if (M <= 0) {
 for (ilpp=2*N-2;ilpp<=2*N-2;ilpp++) {
  R[N-1][N-1] = nrm;
 }
}
if ((M >= 1) && (N >= 2)) {
 for (ilpp=N-1;ilpp<=min(2*N-3,N+M-2);ilpp++) {
  Q[ilpp-N+1][0] = A[ilpp-N+1][0] / R[0][0];
  R[0][ilpp-N+2] = 0.0;
 }
 for (ilpp=2*N-2;ilpp<=N+M-2;ilpp++) {
  Q[ilpp-N+1][0] = A[ilpp-N+1][0] / R[0][0];
 }
 for (ilpp=N+M-1;ilpp<=2*N-3;ilpp++) {
  R[0][ilpp-N+2] = 0.0;
 }
}
if ((M >= 1) && (N == 2)) {
 for (t1=0;t1<=M-1;t1++) {
  for (ilpp=t1+2;ilpp<=t1+2;ilpp++) {
   R[0][1] += Q[t1][0] * A[t1][1];
  }
 }
```

```
}
if ((M >= 1) && (N >= 3)) {
 for (t1=0;t1<=M-1;t1++) {
  for (ilpp=t1+N;ilpp<=t1+2*N-2;ilpp++) {
   R[0][-t1+ilpp-N+1] += Q[t1][0] * A[t1][-t1+ilpp-N+1];
  }
 }
}
if (M >= 1) {
 for (t0=5;t0<=5*N-7;t0++) {
  for (ilpp=(t0+5*N-7)/5;ilpp<=(t0+5*N-7)/5;ilpp++) {
   if ((t0+3)%5 == 0) {
    R[(t0-2)/5][(t0-2)/5] = nrm;
   }
  }
  for (ilpp=(t0+5*N-6)/5;ilpp<=(t0+5*N-6)/5;ilpp++) {
   if ((t0+4)%5 == 0) {
    nrm += A[0][(t0-1)/5] * A[0][(t0-1)/5];
   }
  }
  for (ilpp=(t0+5*N-8)/5;ilpp<=(t0+5*N-8)/5;ilpp++) {
   if ((t0+2)%5 == 0) {
    Q[0][(t0-3)/5] = A[0][(t0-3)/5] / R[(t0-3)/5][(t0-3)/5];
   }
   if ((t0+2)%5 == 0) {
    R[(t0-3)/5][(t0+2)/5] = 0.0;
   }
  }
  if (M >= 2) {
   for (ilpp=(t0+5*N-5)/5;ilpp<=(t0+5*N-5)/5;ilpp++) {
    if (t0%5 == 0) {
     A[0][t0/5] = A[0][t0/5] - Q[0][(t0-5)/5] * R[(t0-5)/5][t0/5];
    }
    if (t0%5 == 0) {
     nrm = 0.0;
    }
   }
  }
  if (M == 1) {
   for (ilpp=(t0+5*N-5)/5;ilpp<=(t0+5*N-5)/5;ilpp++) {
    if (t0%5 == 0) {
     A[0][t0/5] = A[0][t0/5] - Q[0][(t0-5)/5] * R[(t0-5)/5][t0/5];
    }
    if (t0%5 == 0) {
     nrm = 0.0;
    }
   }
  }
  for (ilpp=ceild(t0+5*N-4,5);ilpp<=min(floord(t0+5*N+5*M-13,5),2*N-3);ilpp++) {
   if (t0%5 == 0) {
    for (j=ceild(t0,5);j<=ilpp-N+1;j++) {
     A[ilpp-j-N+1][j] = A[ilpp-j-N+1][j] - Q[ilpp-j-N+1][(t0-5)/5] * R[(t0-5)/5][j];
    }
   }
   if ((t0+1)%5 == 0) {
    R[(t0-4)/5][ilpp-N+1] += Q[0][(t0-4)/5] * A[0][ilpp-N+1];
   }
   if ((t0+2)%5 == 0) {
    Q[(-t0+5*ilpp-5*N+8)/5][(t0-3)/5] = A[(-t0+5*ilpp-5*N+8)/5][(t0-3)/5] / R[(t0-3)/5][(t0-3)/5];
    R[(t0-3)/5][ilpp-N+2] = 0.0;
   }
```

```
  }
  if (t0 >= 5*N-5*M+3) {
   for (ilpp=2*N-2;ilpp<=2*N-2;ilpp++) {
    if (t0%5 == 0) {
     for (j=ceild(t0,5);j<=N-1;j++) {
      A[-j+N-1][j] = A[-j+N-1][j] - Q[-j+N-1][(t0-5)/5] * R[(t0-5)/5][j];
     }
    }
    if ((t0+1)%5 == 0) {
     R[(t0-4)/5][N-1] += Q[0][(t0-4)/5] * A[0][N-1];
    }
    if ((t0+2)%5 == 0) {
     Q[(-t0+5*N-2)/5][(t0-3)/5] = A[(-t0+5*N-2)/5][(t0-3)/5] / R[(t0-3)/5][(t0-3)/5];
    }
   }
  }
  for (ilpp=2*N-1;ilpp<=floord(t0+5*N+5*M-13,5);ilpp++) {
   if (t0%5 == 0) {
    for (j=ceild(t0,5);j<=N-1;j++) {
     A[ilpp-j-N+1][j] = A[ilpp-j-N+1][j] - Q[ilpp-j-N+1][(t0-5)/5] * R[(t0-5)/5][j];
    }
   }
   if ((t0+2)%5 == 0) {
    Q[(-t0+5*ilpp-5*N+8)/5][(t0-3)/5] = A[(-t0+5*ilpp-5*N+8)/5][(t0-3)/5] / R[(t0-3)/5][(t0-3)/5];
   }
  }
  for (ilpp=max(ceild(t0+5*N-4,5),ceild(t0+5*N+5*M-12,5));ilpp<=2*N-3;ilpp++) {
   if (t0%5 == 0) {
    for (j=ilpp-N-M+2;j<=ilpp-N+1;j++) {
     A[ilpp-j-N+1][j] = A[ilpp-j-N+1][j] - Q[ilpp-j-N+1][(t0-5)/5] * R[(t0-5)/5][j];
    }
   }
   if ((t0+1)%5 == 0) {
    R[(t0-4)/5][ilpp-N+1] += Q[0][(t0-4)/5] * A[0][ilpp-N+1];
   }
   if ((t0+2)%5 == 0) {
    R[(t0-3)/5][ilpp-N+2] = 0.0;
   }
  }
  if (t0 <= 5*N-5*M+2) {
   for (ilpp=2*N-2;ilpp<=2*N-2;ilpp++) {
    if (t0%5 == 0) {
     for (j=N-M;j<=N-1;j++) {
      A[-j+N-1][j] = A[-j+N-1][j] - Q[-j+N-1][(t0-5)/5] * R[(t0-5)/5][j];
     }
    }
    if ((t0+1)%5 == 0) {
     R[(t0-4)/5][N-1] += Q[0][(t0-4)/5] * A[0][N-1];
    }
   }
  }
  for (ilpp=max(ceild(t0+5*N+5*M-12,5),2*N-1);ilpp<=2*N+M-3;ilpp++) {
   if (t0%5 == 0) {
    for (j=ilpp-N-M+2;j<=N-1;j++) {
     A[ilpp-j-N+1][j] = A[ilpp-j-N+1][j] - Q[ilpp-j-N+1][(t0-5)/5] * R[(t0-5)/5][j];
    }
   }
  }
  for (t1=1;t1<=M-1;t1++) {
   for (ilpp=(t0+5*N-6)/5;ilpp<=(t0+5*N-6)/5;ilpp++) {
    if ((t0+4)%5 == 0) {
```

```
      nrm += A[t1][(t0-1)/5] * A[t1][(t0-1)/5];
     }
    }
    for (ilpp=ceild(t0+5*t1+5*N-4,5);ilpp<=t1+2*N-2;ilpp++) {
     if ((t0+1)%5 == 0) {
      R[(t0-4)/5][-t1+ilpp-N+1] += Q[t1][(t0-4)/5] * A[t1][-t1+ilpp-N+1];
     }
    }
   }
  }
 }
 if ((M >= 1) && (N >= 3)) {
  for (t1=0;t1<=M-1;t1++) {
   for (ilpp=t1+2*N-2;ilpp<=t1+2*N-2;ilpp++) {
    R[N-2][N-1] += Q[t1][N-2] * A[t1][N-1];
   }
  }
 }
 if ((M >= 1) && (N >= 2)) {
  for (ilpp=2*N-2;ilpp<=2*N-2;ilpp++) {
   A[0][N-1] = A[0][N-1] - Q[0][N-2] * R[N-2][N-1];
   nrm = 0.0;
  }
  for (ilpp=2*N-1;ilpp<=2*N+M-3;ilpp++) {
   A[ilpp-2*N+2][N-1] = A[ilpp-2*N+2][N-1] - Q[ilpp-2*N+2][N-2] * R[N-2][N-1];
  }
 }
 if ((M >= 1) && (N >= 2)) {
  for (t1=0;t1<=M-1;t1++) {
   for (ilpp=2*N-2;ilpp<=2*N-2;ilpp++) {
    nrm += A[t1][N-1] * A[t1][N-1];
   }
  }
 }
 if ((M >= 1) && (N >= 2)) {
  for (ilpp=2*N-2;ilpp<=2*N-2;ilpp++) {
   R[N-1][N-1] = nrm;
  }
 }
 if (M >= 1) {
  for (ilpp=2*N-2;ilpp<=2*N+M-3;ilpp++) {
   Q[ilpp-2*N+2][N-1] = A[ilpp-2*N+2][N-1] / R[N-1][N-1];
  }
 }
}
```

Листинг E.5 gramschmidt_ilp_sync (параллельный вариант ilpy с директивами OpenMP, синхронный параллелизм) на языке C

```
void gramschmidt_ilp_sync(int M, int N, double** A, double** Q, double** R, double* nrm) {
  #pragma omp parallel
  {
    if (N >= 1) {
      if (M >= 1) {
        #pragma omp master
        for (int ilpp=N-1;ilpp<=N-1;ilpp++) {
          *nrm = 0.0;
        }
      }
      if (M <= 0) {
```

```
      #pragma omp master
      for (int ilpp=N-1;ilpp<=N-1;ilpp++) {
        *nrm = 0.0;
      }
    }
    if ((M >= 1) && (N == 1)) {
      for (int t1=0;t1<=M-1;t1++) {
        #pragma omp master
        for (int ilpp=0;ilpp<=0;ilpp++) {
          *nrm += A[t1][0] * A[t1][0];
        }
      }
    }
    if ((M >= 1) && (N == 1)) {
      #pragma omp master
      for (int ilpp=0;ilpp<=0;ilpp++) {
        R[0][0] = sqrt(*nrm);
      }
    }
    if ((M >= 1) && (N >= 2)) {
      for (int t1=0;t1<=M-1;t1++) {
        #pragma omp master
        for (int ilpp=N-1;ilpp<=N-1;ilpp++) {
          *nrm += A[t1][0] * A[t1][0];
        }
      }
    }
    if ((M <= 0) && (N >= 2)) {
      #pragma omp master
      for (int ilpp=N-1;ilpp<=N-1;ilpp++) {
        R[0][0] = sqrt(*nrm);
      }
    }
    if ((M >= 1) && (N >= 2)) {
      #pragma omp master
      for (int ilpp=N-1;ilpp<=N-1;ilpp++) {
        R[0][0] = sqrt(*nrm);
      }
    }
    if (M <= 0) {
      for (int t0=3;t0<=5*N-7;t0++) {
        #pragma omp master
        for (int ilpp=(t0+5*N-7)/5;ilpp<=(t0+5*N-7)/5;ilpp++) {
          if ((t0+3)%5 == 0) {
            R[(t0-2)/5][(t0-2)/5] = sqrt(*nrm);
          }
        }
        #pragma omp master
        for (int ilpp=(t0+5*N-5)/5;ilpp<=(t0+5*N-5)/5;ilpp++) {
          if (t0%5 == 0) {
            *nrm = 0.0;
          }
        }
        if ((t0+2)%5 == 0) {
          #pragma omp barrier
          #pragma omp for
          for (int ilpp=ceild(t0+5*N-8,5);ilpp<=2*N-3;ilpp++) {
            R[(t0-3)/5][ilpp-N+2] = 0.0;
          }
        }
      }
```

```
    }
    if ((M <= 0) && (N >= 2)) {
      #pragma omp master
      for (int ilpp=2*N-2;ilpp<=2*N-2;ilpp++) {
        *nrm = 0.0;
      }
    }
    if (M <= 0) {
      #pragma omp master
      for (int ilpp=2*N-2;ilpp<=2*N-2;ilpp++) {
        R[N-1][N-1] = sqrt(*nrm);
      }
    }
    if ((M >= 1) && (N >= 2)) {
      #pragma omp barrier
      #pragma omp for
      for (int ilpp=N-1;ilpp<=min(2*N-3,N+M-2);ilpp++) {
        Q[ilpp-N+1][0] = A[ilpp-N+1][0] / R[0][0];
        R[0][ilpp-N+2] = 0.0;
      }
      #pragma omp barrier
      #pragma omp for
      for (int ilpp=2*N-2;ilpp<=N+M-2;ilpp++) {
        Q[ilpp-N+1][0] = A[ilpp-N+1][0] / R[0][0];
      }
      #pragma omp barrier
      #pragma omp for
      for (int ilpp=N+M-1;ilpp<=2*N-3;ilpp++) {
        R[0][ilpp-N+2] = 0.0;
      }
    }
    if ((M >= 1) && (N == 2)) {
      for (int t1=0;t1<=M-1;t1++) {
        #pragma omp master
        for (int ilpp=t1+2;ilpp<=t1+2;ilpp++) {
          R[0][1] += Q[t1][0] * A[t1][1];
        }
      }
    }
    if ((M >= 1) && (N >= 3)) {
      for (int t1=0;t1<=M-1;t1++) {
        #pragma omp barrier
        #pragma omp for
        for (int ilpp=t1+N;ilpp<=t1+2*N-2;ilpp++) {
          R[0][-t1+ilpp-N+1] += Q[t1][0] * A[t1][-t1+ilpp-N+1];
        }
      }
    }
    if (M >= 1) {
      for (int t0=5;t0<=5*N-7;t0++) {
        #pragma omp master
        for (int ilpp=(t0+5*N-7)/5;ilpp<=(t0+5*N-7)/5;ilpp++) {
          if ((t0+3)%5 == 0) {
            R[(t0-2)/5][(t0-2)/5] = sqrt(*nrm);
          }
        }
        #pragma omp master
        for (int ilpp=(t0+5*N-6)/5;ilpp<=(t0+5*N-6)/5;ilpp++) {
          if ((t0+4)%5 == 0) {
            *nrm += A[0][(t0-1)/5] * A[0][(t0-1)/5];
          }
```

```
        }
135     #pragma omp master
        for (int ilpp=(t0+5*N-8)/5;ilpp<=(t0+5*N-8)/5;ilpp++) {
          if ((t0+2)%5 == 0) {
            Q[0][(t0-3)/5] = A[0][(t0-3)/5] / R[(t0-3)/5][(t0-3)/5];
          }
140       if ((t0+2)%5 == 0) {
            R[(t0-3)/5][(t0+2)/5] = 0.0;
          }
        }
        if (M >= 2) {
145       #pragma omp master
          for (int ilpp=(t0+5*N-5)/5;ilpp<=(t0+5*N-5)/5;ilpp++) {
            if (t0%5 == 0) {
              A[0][t0/5] = A[0][t0/5] - Q[0][(t0-5)/5] * R[(t0-5)/5][t0/5];
            }
150         if (t0%5 == 0) {
              *nrm = 0.0;
            }
          }
        }
155     if (M == 1) {
          #pragma omp master
          for (int ilpp=(t0+5*N-5)/5;ilpp<=(t0+5*N-5)/5;ilpp++) {
            if (t0%5 == 0) {
              A[0][t0/5] = A[0][t0/5] - Q[0][(t0-5)/5] * R[(t0-5)/5][t0/5];
160         }
            if (t0%5 == 0) {
              *nrm = 0.0;
            }
          }
165     }
        if (t0%5 == 0) {
          #pragma omp barrier
          #pragma omp for
          for (int ilpp=ceild(t0+5*N-4,5);ilpp<=min(floord(t0+5*N+5*M-13,5),2*N-3);ilpp++) {
170         for (int j=ceild(t0,5);j<=ilpp-N+1;j++) {
              A[ilpp-j-N+1][j] = A[ilpp-j-N+1][j] - Q[ilpp-j-N+1][(t0-5)/5] * R[(t0-5)/5][j];
            }
          }
        }
175     if ((t0+1)%5 == 0) {
          #pragma omp barrier
          #pragma omp for
          for (int ilpp=ceild(t0+5*N-4,5);ilpp<=min(floord(t0+5*N+5*M-13,5),2*N-3);ilpp++) {
            R[(t0-4)/5][ilpp-N+1] += Q[0][(t0-4)/5] * A[0][ilpp-N+1];
180       }
        }
        if ((t0+2)%5 == 0) {
          #pragma omp barrier
          #pragma omp for
185       for (int ilpp=ceild(t0+5*N-4,5);ilpp<=min(floord(t0+5*N+5*M-13,5),2*N-3);ilpp++) {
            Q[(-t0+5*ilpp-5*N+8)/5][(t0-3)/5] = A[(-t0+5*ilpp-5*N+8)/5][(t0-3)/5] /
↪ R[(t0-3)/5][(t0-3)/5];
            R[(t0-3)/5][ilpp-N+2] = 0.0;
          }
        }
190     /*#pragma omp barrier
        #pragma omp for
        for (int ilpp=ceild(t0+5*N-4,5);ilpp<=min(floord(t0+5*N+5*M-13,5),2*N-3);ilpp++) {
          if (t0%5 == 0) {
```

```
            for (int j=ceild(t0,5);j<=ilpp-N+1;j++) {
              A[ilpp-j-N+1][j] = A[ilpp-j-N+1][j] - Q[ilpp-j-N+1][(t0-5)/5] * R[(t0-5)/5][j];
            }
          }
          if ((t0+1)%5 == 0) {
            R[(t0-4)/5][ilpp-N+1] += Q[0][(t0-4)/5] * A[0][ilpp-N+1];
          }
          if ((t0+2)%5 == 0) {
            Q[(-t0+5*ilpp-5*N+8)/5][(t0-3)/5] = A[(-t0+5*ilpp-5*N+8)/5][(t0-3)/5] /
↪ R[(t0-3)/5][(t0-3)/5];
            R[(t0-3)/5][ilpp-N+2] = 0.0;
          }
        }*/
        if (t0 >= 5*N-5*M+3) {
          #pragma omp master
          for (int ilpp=2*N-2;ilpp<=2*N-2;ilpp++) {
            if (t0%5 == 0) {
              for (int j=ceild(t0,5);j<=N-1;j++) {
                A[-j+N-1][j] = A[-j+N-1][j] - Q[-j+N-1][(t0-5)/5] * R[(t0-5)/5][j];
              }
            }
            if ((t0+1)%5 == 0) {
              R[(t0-4)/5][N-1] += Q[0][(t0-4)/5] * A[0][N-1];
            }
            if ((t0+2)%5 == 0) {
              Q[(-t0+5*N-2)/5][(t0-3)/5] = A[(-t0+5*N-2)/5][(t0-3)/5] / R[(t0-3)/5][(t0-3)/5];
            }
          }
        }
        if (t0%5 == 0) {
          #pragma omp barrier
          #pragma omp for
          for (int ilpp=2*N-1;ilpp<=floord(t0+5*N+5*M-13,5);ilpp++) {
            for (int j=ceild(t0,5);j<=N-1;j++) {
              A[ilpp-j-N+1][j] = A[ilpp-j-N+1][j] - Q[ilpp-j-N+1][(t0-5)/5] * R[(t0-5)/5][j];
            }
          }
        }
        if ((t0+2)%5 == 0) {
          #pragma omp barrier
          #pragma omp for
          for (int ilpp=2*N-1;ilpp<=floord(t0+5*N+5*M-13,5);ilpp++) {
            Q[(-t0+5*ilpp-5*N+8)/5][(t0-3)/5] = A[(-t0+5*ilpp-5*N+8)/5][(t0-3)/5] /
↪ R[(t0-3)/5][(t0-3)/5];
          }
        }
        /*#pragma omp barrier
        #pragma omp for
        for (int ilpp=2*N-1;ilpp<=floord(t0+5*N+5*M-13,5);ilpp++) {
          if (t0%5 == 0) {
            for (int j=ceild(t0,5);j<=N-1;j++) {
              A[ilpp-j-N+1][j] = A[ilpp-j-N+1][j] - Q[ilpp-j-N+1][(t0-5)/5] * R[(t0-5)/5][j];
            }
          }
          if ((t0+2)%5 == 0) {
            Q[(-t0+5*ilpp-5*N+8)/5][(t0-3)/5] = A[(-t0+5*ilpp-5*N+8)/5][(t0-3)/5] /
↪ R[(t0-3)/5][(t0-3)/5];
          }
        }*/
        if (t0%5 == 0) {
          #pragma omp barrier
```

```
    #pragma omp for
    for (int ilpp=max(ceild(t0+5*N-4,5),ceild(t0+5*N+5*M-12,5));ilpp<=2*N-3;ilpp++) {
      for (int j=ilpp-N-M+2;j<=ilpp-N+1;j++) {
        A[ilpp-j-N+1][j] = A[ilpp-j-N+1][j] - Q[ilpp-j-N+1][(t0-5)/5] * R[(t0-5)/5][j];
      }
    }
  }
  if ((t0+1)%5 == 0) {
    #pragma omp barrier
    #pragma omp for
    for (int ilpp=max(ceild(t0+5*N-4,5),ceild(t0+5*N+5*M-12,5));ilpp<=2*N-3;ilpp++) {
      R[(t0-4)/5][ilpp-N+1] += Q[0][(t0-4)/5] * A[0][ilpp-N+1];
    }
  }
  if ((t0+2)%5 == 0) {
    #pragma omp barrier
    #pragma omp for
    for (int ilpp=max(ceild(t0+5*N-4,5),ceild(t0+5*N+5*M-12,5));ilpp<=2*N-3;ilpp++) {
      R[(t0-3)/5][ilpp-N+2] = 0.0;
    }
  }
  /*#pragma omp barrier
  #pragma omp for
  for (int ilpp=max(ceild(t0+5*N-4,5),ceild(t0+5*N+5*M-12,5));ilpp<=2*N-3;ilpp++) {
    if (t0%5 == 0) {
      for (int j=ilpp-N-M+2;j<=ilpp-N+1;j++) {
        A[ilpp-j-N+1][j] = A[ilpp-j-N+1][j] - Q[ilpp-j-N+1][(t0-5)/5] * R[(t0-5)/5][j];
      }
    }
    if ((t0+1)%5 == 0) {
      R[(t0-4)/5][ilpp-N+1] += Q[0][(t0-4)/5] * A[0][ilpp-N+1];
    }
    if ((t0+2)%5 == 0) {
      R[(t0-3)/5][ilpp-N+2] = 0.0;
    }
  }*/
  if (t0 <= 5*N-5*M+2) {
    #pragma omp master
    for (int ilpp=2*N-2;ilpp<=2*N-2;ilpp++) {
      if (t0%5 == 0) {
        for (int j=N-M;j<=N-1;j++) {
          A[-j+N-1][j] = A[-j+N-1][j] - Q[-j+N-1][(t0-5)/5] * R[(t0-5)/5][j];
        }
      }
      if ((t0+1)%5 == 0) {
        R[(t0-4)/5][N-1] += Q[0][(t0-4)/5] * A[0][N-1];
      }
    }
  }
  if (t0%5 == 0) {
    #pragma omp barrier
    #pragma omp for
    for (int ilpp=max(ceild(t0+5*N+5*M-12,5),2*N-1);ilpp<=2*N+M-3;ilpp++) {
      for (int j=ilpp-N-M+2;j<=N-1;j++) {
        A[ilpp-j-N+1][j] = A[ilpp-j-N+1][j] - Q[ilpp-j-N+1][(t0-5)/5] * R[(t0-5)/5][j];
      }
    }
  }
  for (int t1=1;t1<=M-1;t1++) {
    #pragma omp master
    for (int ilpp=(t0+5*N-6)/5;ilpp<=(t0+5*N-6)/5;ilpp++) {
```

```
              if ((t0+4)%5 == 0) {
                *nrm += A[t1][(t0-1)/5] * A[t1][(t0-1)/5];
              }
            }
            if ((t0+1)%5 == 0) {
              #pragma omp barrier
              #pragma omp for
              for (int ilpp=ceild(t0+5*t1+5*N-4,5);ilpp<=t1+2*N-2;ilpp++) {
                R[(t0-4)/5][-t1+ilpp-N+1] += Q[t1][(t0-4)/5] * A[t1][-t1+ilpp-N+1];
              }
            }
          }
        }
      }
      if ((M >= 1) && (N >= 3)) {
        for (int t1=0;t1<=M-1;t1++) {
          #pragma omp master
          for (int ilpp=t1+2*N-2;ilpp<=t1+2*N-2;ilpp++) {
            R[N-2][N-1] += Q[t1][N-2] * A[t1][N-1];
          }
        }
      }
      if ((M >= 1) && (N >= 2)) {
        #pragma omp master
        for (int ilpp=2*N-2;ilpp<=2*N-2;ilpp++) {
          A[0][N-1] = A[0][N-1] - Q[0][N-2] * R[N-2][N-1];
          *nrm = 0.0;
        }
        #pragma omp barrier
        #pragma omp for
        for (int ilpp=2*N-1;ilpp<=2*N+M-3;ilpp++) {
          A[ilpp-2*N+2][N-1] = A[ilpp-2*N+2][N-1] - Q[ilpp-2*N+2][N-2] * R[N-2][N-1];
        }
      }
      if ((M >= 1) && (N >= 2)) {
        for (int t1=0;t1<=M-1;t1++) {
          #pragma omp master
          for (int ilpp=2*N-2;ilpp<=2*N-2;ilpp++) {
            *nrm += A[t1][N-1] * A[t1][N-1];
          }
        }
      }
      if ((M >= 1) && (N >= 2)) {
        #pragma omp master
        for (int ilpp=2*N-2;ilpp<=2*N-2;ilpp++) {
          R[N-1][N-1] = sqrt(*nrm);
        }
      }
      if (M >= 1) {
        #pragma omp barrier
        #pragma omp for
        for (int ilpp=2*N-2;ilpp<=2*N+M-3;ilpp++) {
          Q[ilpp-2*N+2][N-1] = A[ilpp-2*N+2][N-1] / R[N-1][N-1];
        }
      }
    }
  }
}
```

Листинг E.6 gramschmidt (параллельный вариант ilpy без конструкций MPI, синхронный параллелизм) на языке C

```
if (N >= 1) {
 if (M >= 1) {
  for (ilpp=0;ilpp<=0;ilpp++) {
   nrm = 0.0;
  }
 }
 if (M <= 0) {
  for (ilpp=0;ilpp<=0;ilpp++) {
   nrm = 0.0;
  }
 }
 if ((M >= 1) && (N == 1)) {
  for (t1=0;t1<=M-1;t1++) {
   for (ilpp=1;ilpp<=1;ilpp++) {
    nrm += A[t1][0] * A[t1][0];
   }
  }
 }
 if ((M >= 1) && (N == 1)) {
  for (ilpp=1;ilpp<=1;ilpp++) {
   R[0][0] = nrm;
  }
 }
 if ((M >= 1) && (N >= 2)) {
  for (t1=0;t1<=M-1;t1++) {
   for (ilpp=1;ilpp<=1;ilpp++) {
    nrm += A[t1][0] * A[t1][0];
   }
  }
 }
 if ((M <= 0) && (N >= 2)) {
  for (ilpp=1;ilpp<=1;ilpp++) {
   R[0][0] = nrm;
  }
 }
 if ((M >= 1) && (N >= 2)) {
  for (ilpp=1;ilpp<=1;ilpp++) {
   R[0][0] = nrm;
  }
 }
 if (M <= 0) {
  for (t0=3;t0<=5*N-7;t0++) {
   for (ilpp=(t0+3)/5;ilpp<=(t0+3)/5;ilpp++) {
    if ((t0+3)%5 == 0) {
     R[(t0-2)/5][(t0-2)/5] = nrm;
    }
   }
   for (ilpp=t0/5;ilpp<=t0/5;ilpp++) {
    if (t0%5 == 0) {
     nrm = 0.0;
    }
   }
   for (ilpp=ceild(t0+7,5);ilpp<=N;ilpp++) {
    if ((t0+2)%5 == 0) {
     R[(t0-3)/5][ilpp-1] = 0.0;
    }
   }
```

```
 }
}
if ((M <= 0) && (N >= 2)) {
 for (ilpp=N-1;ilpp<=N-1;ilpp++) {
  nrm = 0.0;
 }
}
if (M <= 0) {
 for (ilpp=N;ilpp<=N;ilpp++) {
  R[N-1][N-1] = nrm;
 }
}
if ((M >= 1) && (N >= 2)) {
 for (ilpp=0;ilpp<=min(1,M-1);ilpp++) {
  Q[ilpp][0] = A[ilpp][0] / R[0][0];
 }
 for (ilpp=2;ilpp<=min(N,M-1);ilpp++) {
  R[0][ilpp-1] = 0.0;
  Q[ilpp][0] = A[ilpp][0] / R[0][0];
 }
 for (ilpp=N+1;ilpp<=M-1;ilpp++) {
  Q[ilpp][0] = A[ilpp][0] / R[0][0];
 }
 for (ilpp=max(2,M);ilpp<=N;ilpp++) {
  R[0][ilpp-1] = 0.0;
 }
}
if ((M >= 1) && (N == 2)) {
 for (t1=0;t1<=M-1;t1++) {
  for (ilpp=2;ilpp<=2;ilpp++) {
   R[0][1] += Q[t1][0] * A[t1][1];
  }
 }
}
if ((M >= 1) && (N >= 3)) {
 for (t1=0;t1<=M-1;t1++) {
  for (ilpp=2;ilpp<=N;ilpp++) {
   R[0][ilpp-1] += Q[t1][0] * A[t1][ilpp-1];
  }
 }
}
if (M >= 1) {
 for (t0=5;t0<=5*N-7;t0++) {
  for (ilpp=t0/5;ilpp<=t0/5;ilpp++) {
   if (t0%5 == 0) {
    nrm = 0.0;
   }
  }
  for (ilpp=ceild(t0-3,5);ilpp<=min(floord(t0+2,5),floord(t0+5*M-8,5));ilpp++) {
   if ((t0+2)%5 == 0) {
    Q[(-t0+5*ilpp+3)/5][(t0-3)/5] = A[(-t0+5*ilpp+3)/5][(t0-3)/5] / R[(t0-3)/5][(t0-3)/5];
   }
  }
  if (M >= 3) {
   for (ilpp=(t0+5)/5;ilpp<=(t0+5)/5;ilpp++) {
    if (t0%5 == 0) {
     for (k=0;k<=M-1;k++) {
      A[k][t0/5] = A[k][t0/5] - Q[k][(t0-5)/5] * R[(t0-5)/5][t0/5];
     }
    }
   }
```

```
  }
  if (M <= 2) {
   for (ilpp=(t0+5)/5;ilpp<=(t0+5)/5;ilpp++) {
    if (t0%5 == 0) {
     for (k=0;k<=M-1;k++) {
      A[k][t0/5] = A[k][t0/5] - Q[k][(t0-5)/5] * R[(t0-5)/5][t0/5];
     }
    }
   }
  }
  if (M >= 3) {
   for (ilpp=(t0+6)/5;ilpp<=(t0+6)/5;ilpp++) {
    if ((t0+1)%5 == 0) {
     R[(t0-4)/5][(t0+1)/5] += Q[0][(t0-4)/5] * A[0][(t0+1)/5];
    }
   }
  }
  if (M <= 2) {
   for (ilpp=(t0+6)/5;ilpp<=(t0+6)/5;ilpp++) {
    if ((t0+1)%5 == 0) {
     R[(t0-4)/5][(t0+1)/5] += Q[0][(t0-4)/5] * A[0][(t0+1)/5];
    }
   }
  }
  if (M >= 3) {
   for (ilpp=(t0+3)/5;ilpp<=(t0+3)/5;ilpp++) {
    if ((t0+3)%5 == 0) {
     R[(t0-2)/5][(t0-2)/5] = nrm;
    }
   }
  }
  if (M <= 2) {
   for (ilpp=(t0+3)/5;ilpp<=(t0+3)/5;ilpp++) {
    if ((t0+3)%5 == 0) {
     R[(t0-2)/5][(t0-2)/5] = nrm;
    }
   }
  }
  if (M >= 3) {
   for (ilpp=(t0+4)/5;ilpp<=(t0+4)/5;ilpp++) {
    if ((t0+4)%5 == 0) {
     nrm += A[0][(t0-1)/5] * A[0][(t0-1)/5];
    }
   }
  }
  if (M <= 2) {
   for (ilpp=(t0+4)/5;ilpp<=(t0+4)/5;ilpp++) {
    if ((t0+4)%5 == 0) {
     nrm += A[0][(t0-1)/5] * A[0][(t0-1)/5];
    }
   }
  }
  for (ilpp=ceild(t0+7,5);ilpp<=min(floord(t0+5*M-8,5),N);ilpp++) {
   if (t0%5 == 0) {
    for (k=0;k<=M-1;k++) {
     A[k][ilpp-1] = A[k][ilpp-1] - Q[k][(t0-5)/5] * R[(t0-5)/5][ilpp-1];
    }
   }
   if ((t0+1)%5 == 0) {
    R[(t0-4)/5][ilpp-1] += Q[0][(t0-4)/5] * A[0][ilpp-1];
   }
```

```
    if ((t0+2)%5 == 0) {
     Q[(-t0+5*ilpp+3)/5][(t0-3)/5] = A[(-t0+5*ilpp+3)/5][(t0-3)/5] / R[(t0-3)/5][(t0-3)/5];
     R[(t0-3)/5][ilpp-1] = 0.0;
    }
   }
   for (ilpp=N+1;ilpp<=floord(t0+5*M-8,5);ilpp++) {
    if ((t0+2)%5 == 0) {
     Q[(-t0+5*ilpp+3)/5][(t0-3)/5] = A[(-t0+5*ilpp+3)/5][(t0-3)/5] / R[(t0-3)/5][(t0-3)/5];
    }
   }
   for (ilpp=max(ceild(t0+7,5),ceild(t0+5*M-7,5));ilpp<=N;ilpp++) {
    if (t0%5 == 0) {
     for (k=0;k<=M-1;k++) {
      A[k][ilpp-1] = A[k][ilpp-1] - Q[k][(t0-5)/5] * R[(t0-5)/5][ilpp-1];
     }
    }
    if ((t0+1)%5 == 0) {
     R[(t0-4)/5][ilpp-1] += Q[0][(t0-4)/5] * A[0][ilpp-1];
    }
    if ((t0+2)%5 == 0) {
     R[(t0-3)/5][ilpp-1] = 0.0;
    }
   }
   for (t1=1;t1<=M-1;t1++) {
    for (ilpp=(t0+4)/5;ilpp<=(t0+4)/5;ilpp++) {
     if ((t0+4)%5 == 0) {
      nrm += A[t1][(t0-1)/5] * A[t1][(t0-1)/5];
     }
    }
    for (ilpp=ceild(t0+6,5);ilpp<=N;ilpp++) {
     if ((t0+1)%5 == 0) {
      R[(t0-4)/5][ilpp-1] += Q[t1][(t0-4)/5] * A[t1][ilpp-1];
     }
    }
   }
  }
 }
 if ((M >= 1) && (N >= 3)) {
  for (t1=0;t1<=M-1;t1++) {
   for (ilpp=N;ilpp<=N;ilpp++) {
    R[N-2][N-1] += Q[t1][N-2] * A[t1][N-1];
   }
  }
 }
 if ((M >= 1) && (N >= 2)) {
  for (ilpp=N-1;ilpp<=N-1;ilpp++) {
   nrm = 0.0;
  }
  for (ilpp=N;ilpp<=N;ilpp++) {
   for (k=0;k<=M-1;k++) {
    A[k][N-1] = A[k][N-1] - Q[k][N-2] * R[N-2][N-1];
   }
  }
 }
 if ((M >= 1) && (N >= 2)) {
  for (t1=0;t1<=M-1;t1++) {
   for (ilpp=N;ilpp<=N;ilpp++) {
    nrm += A[t1][N-1] * A[t1][N-1];
   }
  }
 }
}
```

```
 if ((M >= 1) && (N >= 2)) {
  for (ilpp=N;ilpp<=N;ilpp++) {
   R[N-1][N-1] = nrm;
  }
245 }
 if (M >= 1) {
  for (ilpp=N-1;ilpp<=N+M-2;ilpp++) {
   Q[ilpp-N+1][N-1] = A[ilpp-N+1][N-1] / R[N-1][N-1];
  }
250 }
}
```

Листинг E.7 gramschmidt_ilp_arrays (параллельный вариант ilpy с конструкциями MPI, синхронный параллелизм) на языке C

```
// arrays placement

#define T_base 1000000

 5 #define eta_A(i0, i1) ((i1) + 1)
#define eta_A_col(i1) eta_A(0, i1)
#define T_A (1 * T_base)

int A_chunk_size;
10 int A_actual_chunk_size;

#define eta_Q(i0, i1) ((i1) + 1)
#define eta_Q_col(i1) eta_Q(0, i1)
#define T_Q (2 * T_base)
15
int Q_chunk_size;
int Q_actual_chunk_size;

#define eta_R(i0, i1) ((i1) + 1)
20 #define eta_R_col(i1) eta_R(0, i1)
#define T_R (3 * T_base)

int R_chunk_size;
int R_actual_chunk_size;
25
// mpi routine

double* buf_Am;
double* buf_Qm;
30 double* buf_Rm;

void gramschmidt_ilp_arrays(int M, int N, double** Am, double** Qm, double** Rm) {
  double nrm;
  if (N >= 1) {
35   if (M >= 1) {
      for (int ilpp=max(lR,0);ilpp<=min(uR,0);ilpp++) {
        nrm = 0.0;
      }
      MPI_Barrier_time(MPI_COMM_WORLD);
40   }
     if (M <= 0) {
      for (int ilpp=max(lR,0);ilpp<=min(uR,0);ilpp++) {
        nrm = 0.0;
      }
45    MPI_Barrier_time(MPI_COMM_WORLD);
```

```
    }
    if ((M >= 1) && (N == 1)) {
      for (int t1=0;t1<=M-1;t1++) {
        for (int ilpp=max(lR,1);ilpp<=min(uR,1);ilpp++) {
          nrm += Am[t1][0] * Am[t1][0];
        }
        MPI_Barrier_time(MPI_COMM_WORLD);
      }
    }
    if ((M >= 1) && (N == 1)) {
      for (int ilpp=max(lR,1);ilpp<=min(uR,1);ilpp++) {
        Rm[0][0] = sqrt(nrm);
      }
      MPI_Barrier_time(MPI_COMM_WORLD);
    }
    if ((M >= 1) && (N >= 2)) {
      for (int t1=0;t1<=M-1;t1++) {
        for (int ilpp=max(lR,1);ilpp<=min(uR,1);ilpp++) {
          nrm += Am[t1][0] * Am[t1][0];
        }
        MPI_Barrier_time(MPI_COMM_WORLD);
      }
    }
    if ((M <= 0) && (N >= 2)) {
      for (int ilpp=max(lR,1);ilpp<=min(uR,1);ilpp++) {
        Rm[0][0] = sqrt(nrm);
      }
      MPI_Barrier_time(MPI_COMM_WORLD);
    }
    if ((M >= 1) && (N >= 2)) {
      for (int ilpp=max(lR,1);ilpp<=min(uR,1);ilpp++) {
        Rm[0][0] = sqrt(nrm);
      }
      MPI_Barrier_time(MPI_COMM_WORLD);
    }
    if (M <= 0) {
      for (int t0=3;t0<=5*N-7;t0++) {
        for (int ilpp=max(lR,(t0+3)/5);ilpp<=min(uR,(t0+3)/5);ilpp++) {
          if ((t0+3)%5 == 0) {
            Rm[(t0-2)/5][(t0-2)/5] = sqrt(nrm);
          }
        }
        MPI_Barrier_time(MPI_COMM_WORLD);
        for (int ilpp=max(lR,t0/5);ilpp<=min(uR,t0/5);ilpp++) {
          if (t0%5 == 0) {
            nrm = 0.0;
          }
        }
        MPI_Barrier_time(MPI_COMM_WORLD);
        for (int ilpp=max(lR,ceild(t0+7,5));ilpp<=min(uR,N);ilpp++) {
          if ((t0+2)%5 == 0) {
            Rm[(t0-3)/5][ilpp-1] = 0.0;
          }
        }
        MPI_Barrier_time(MPI_COMM_WORLD);
      }
    }
    if ((M <= 0) && (N >= 2)) {
      for (int ilpp=max(lR,N-1);ilpp<=min(uR,N-1);ilpp++) {
        nrm = 0.0;
      }
```

```
      MPI_Barrier_time(MPI_COMM_WORLD);
    }
    if (M <= 0) {
      for (int ilpp=max(lR,N);ilpp<=min(uR,N);ilpp++) {
        Rm[N-1][N-1] = sqrt(nrm);
      }
      MPI_Barrier_time(MPI_COMM_WORLD);
    }
    if ((M >= 1) && (N >= 2)) {
      // Am[ilpp][0] in Qm[ilpp][0] = Am[ilpp][0] / Rm[0][0]
      col_Q_read_vp(1, 0, 0, min(1,M-1), 0, buf_Am, Am, eta_A, T_A);
      //
      // Rm[0][0] in Qm[ilpp][0] = Am[ilpp][0] / Rm[0][0]
      line_Q_read_vp(0, 0, 0, min(1,M-1), Rm[0], eta_R_col, T_R + 0);
      //
      for (int ilpp=max(lR,0);ilpp<=min(uR,min(1,M-1));ilpp++) {
        Qm[ilpp][0] = Am[ilpp][0] / Rm[0][0];
      }
      MPI_Barrier_time(MPI_COMM_WORLD);
      // Qm[ilpp][0] in Qm[ilpp][0] = Am[ilpp][0] / Rm[0][0]
      col_Q_write_vp(1, 0, 0, min(1,M-1), 0, buf_Qm, Qm, eta_Q, T_Q);
      //
      // Am[ilpp][0] in Qm[ilpp][0] = Am[ilpp][0] / Rm[0][0]
      col_Q_read_vp(1, 0, 2, min(N,M-1), 0, buf_Am, Am, eta_A, T_A);
      //
      // Rm[0][0] in Qm[ilpp][0] = Am[ilpp][0] / Rm[0][0]
      line_Q_read_vp(0, 0, 2, min(N,M-1), Rm[0], eta_R_col, T_R + 0);
      //
      for (int ilpp=max(lR,2);ilpp<=min(uR,min(N,M-1));ilpp++) {
        Rm[0][ilpp-1] = 0.0;
        Qm[ilpp][0] = Am[ilpp][0] / Rm[0][0];
      }
      MPI_Barrier_time(MPI_COMM_WORLD);
      // Qm[ilpp][0] in Qm[ilpp][0] = Am[ilpp][0] / Rm[0][0]
      col_Q_write_vp(1, 0, 2, min(N,M-1), 0, buf_Qm, Qm, eta_Q, T_Q);
      //
      // Am[ilpp][0] in Qm[ilpp][0] = Am[ilpp][0] / Rm[0][0]
      col_Q_read_vp(1, 0, N+1, M-1, 0, buf_Am, Am, eta_A, T_A);
      //
      // Rm[0][0] in Qm[ilpp][0] = Am[ilpp][0] / Rm[0][0]
      line_Q_read_vp(0, 0, N+1, M-1, Rm[0], eta_R_col, T_R + 0);
      //
      for (int ilpp=max(lR,N+1);ilpp<=min(uR,M-1);ilpp++) {
        Qm[ilpp][0] = Am[ilpp][0] / Rm[0][0];
      }
      MPI_Barrier_time(MPI_COMM_WORLD);
      // Qm[ilpp][0] in Qm[ilpp][0] = Am[ilpp][0] / Rm[0][0]
      col_Q_write_vp(1, 0, N+1, M-1, 0, buf_Qm, Qm, eta_Q, T_Q);
      //
      for (int ilpp=max(lR,max(2,M));ilpp<=min(uR,N);ilpp++) {
        Rm[0][ilpp-1] = 0.0;
      }
      MPI_Barrier_time(MPI_COMM_WORLD);
    }
    if ((M >= 1) && (N == 2)) {
      for (int t1=0;t1<=M-1;t1++) {
        // Qm[t1][0] in Rm[0][1] += Qm[t1][0] * Am[t1][1]
        line_Q_read_vp(0, 0, 2, 2, Qm[t1], eta_Q_col, T_Q + (t1));
        //
        for (int ilpp=max(lR,2);ilpp<=min(uR,2);ilpp++) {
          Rm[0][1] += Qm[t1][0] * Am[t1][1];
```

```
      }
      MPI_Barrier_time(MPI_COMM_WORLD);
    }
  }
  if ((M >= 1) && (N >= 3)) {
    for (int t1=0;t1<=M-1;t1++) {
      // Qm[t1][0] in Rm[0][ilpp-1] += Qm[t1][0] * Am[t1][ilpp-1]
      line_Q_read_vp(0, 0, 2, N, Qm[t1], eta_Q_col, T_Q + (t1));
      //
      for (int ilpp=max(lR,2);ilpp<=min(uR,N);ilpp++) {
        Rm[0][ilpp-1] += Qm[t1][0] * Am[t1][ilpp-1];
      }
      MPI_Barrier_time(MPI_COMM_WORLD);
    }
  }
  if (M >= 1) {
    for (int t0=5;t0<=5*N-7;t0++) {
      for (int ilpp=max(lR,t0/5);ilpp<=min(uR,t0/5);ilpp++) {
        if (t0%5 == 0) {
          nrm = 0.0;
        }
      }
      MPI_Barrier_time(MPI_COMM_WORLD);
      if ((t0+2)%5 == 0) {
        // Am[(-t0+5*ilpp+3)/5][(t0-3)/5] in Qm[(-t0+5*ilpp+3)/5][(t0-3)/5] =
↪ Am[(-t0+5*ilpp+3)/5][(t0-3)/5] / Rm[(t0-3)/5][(t0-3)/5]
        col_Q_read_vp(1, (-t0+3)/5, ceild(t0-3,5), min(floord(t0+2,5),floord(t0+5*M-8,5)),
↪ (t0-3)/5, buf_Am, Am, eta_A, T_A);
        //
        // Rm[(t0-3)/5][(t0-3)/5] in Qm[(-t0+5*ilpp+3)/5][(t0-3)/5] =
↪ Am[(-t0+5*ilpp+3)/5][(t0-3)/5] / Rm[(t0-3)/5][(t0-3)/5]
        line_Q_read_vp(0, (t0-3)/5, ceild(t0-3,5), min(floord(t0+2,5),floord(t0+5*M-8,5)),
↪ Rm[(t0-3)/5], eta_R_col, T_R + ((t0-3)/5));
        //
        for (int
↪ ilpp=max(lR,ceild(t0-3,5));ilpp<=min(uR,min(floord(t0+2,5),floord(t0+5*M-8,5)));ilpp++) {
          Qm[(-t0+5*ilpp+3)/5][(t0-3)/5] = Am[(-t0+5*ilpp+3)/5][(t0-3)/5] / Rm[(t0-3)/5][(t0-3)/5];
        }
        MPI_Barrier_time(MPI_COMM_WORLD);
        // Qm[(-t0+5*ilpp+3)/5][(t0-3)/5] in Qm[(-t0+5*ilpp+3)/5][(t0-3)/5] =
↪ Am[(-t0+5*ilpp+3)/5][(t0-3)/5] / Rm[(t0-3)/5][(t0-3)/5]
        col_Q_write_vp(1, (-t0+3)/5, ceild(t0-3,5), min(floord(t0+2,5),floord(t0+5*M-8,5)),
↪ (t0-3)/5, buf_Qm, Qm, eta_Q, T_Q);
        //
      }
      if (M >= 3) {
        if (t0%5 == 0) {
          // Qm[k][(t0-5)/5] in Am[k][t0/5] = Am[k][t0/5] - Qm[k][(t0-5)/5] * Rm[(t0-5)/5][t0/5]
          for (int k=0;k<=M-1;k++) {
            line_Q_read_vp(0, (t0-5)/5, (t0+5)/5, (t0+5)/5, Qm[k], eta_Q_col, T_Q + (k));
          }
          //
          for (int ilpp=max(lR,(t0+5)/5);ilpp<=min(uR,(t0+5)/5);ilpp++) {
            for (int k=0;k<=M-1;k++) {
              Am[k][t0/5] = Am[k][t0/5] - Qm[k][(t0-5)/5] * Rm[(t0-5)/5][t0/5];
            }
          }
          MPI_Barrier_time(MPI_COMM_WORLD);
        }
      }
      if (M <= 2) {
```

```
          if (t0%5 == 0) {
            // Qm[k][(t0-5)/5] in Am[k][t0/5] = Am[k][t0/5] - Qm[k][(t0-5)/5] * Rm[(t0-5)/5][t0/5]
            for (int k=0;k<=M-1;k++) {
              line_Q_read_vp(0, (t0-5)/5, (t0+5)/5, (t0+5)/5, Qm[k], eta_Q_col, T_Q + (k));
            }
            //
            for (int ilpp=max(lR,(t0+5)/5);ilpp<=min(uR,(t0+5)/5);ilpp++) {
              for (int k=0;k<=M-1;k++) {
                Am[k][t0/5] = Am[k][t0/5] - Qm[k][(t0-5)/5] * Rm[(t0-5)/5][t0/5];
              }
            }
            MPI_Barrier_time(MPI_COMM_WORLD);
          }
        }
        if (M >= 3) {
          if ((t0+1)%5 == 0) {
            // Qm[0][(t0-4)/5] in Rm[(t0-4)/5][(t0+1)/5] += Qm[0][(t0-4)/5] * Am[0][(t0+1)/5]
            line_Q_read_vp(0, (t0-4)/5, (t0+6)/5, (t0+6)/5, Qm[0], eta_Q_col, T_Q + 0);
            //
            for (int ilpp=max(lR,(t0+6)/5);ilpp<=min(uR,(t0+6)/5);ilpp++) {
              Rm[(t0-4)/5][(t0+1)/5] += Qm[0][(t0-4)/5] * Am[0][(t0+1)/5];
            }
            MPI_Barrier_time(MPI_COMM_WORLD);
          }
        }
        if (M <= 2) {
          if ((t0+1)%5 == 0) {
            // Qm[0][(t0-4)/5] in Rm[(t0-4)/5][(t0+1)/5] += Qm[0][(t0-4)/5] * Am[0][(t0+1)/5]
            line_Q_read_vp(0, (t0-4)/5, (t0+6)/5, (t0+6)/5, Qm[0], eta_Q_col, T_Q + 0);
            //
            for (int ilpp=max(lR,(t0+6)/5);ilpp<=min(uR,(t0+6)/5);ilpp++) {
              Rm[(t0-4)/5][(t0+1)/5] += Qm[0][(t0-4)/5] * Am[0][(t0+1)/5];
            }
            MPI_Barrier_time(MPI_COMM_WORLD);
          }
        }
        if (M >= 3) {
          for (int ilpp=max(lR,(t0+3)/5);ilpp<=min(uR,(t0+3)/5);ilpp++) {
            if ((t0+3)%5 == 0) {
              Rm[(t0-2)/5][(t0-2)/5] = sqrt(nrm);
            }
          }
          MPI_Barrier_time(MPI_COMM_WORLD);
        }
        if (M <= 2) {
          for (int ilpp=max(lR,(t0+3)/5);ilpp<=min(uR,(t0+3)/5);ilpp++) {
            if ((t0+3)%5 == 0) {
              Rm[(t0-2)/5][(t0-2)/5] = sqrt(nrm);
            }
          }
          MPI_Barrier_time(MPI_COMM_WORLD);
        }
        if (M >= 3) {
          for (int ilpp=max(lR,(t0+4)/5);ilpp<=min(uR,(t0+4)/5);ilpp++) {
            if ((t0+4)%5 == 0) {
              nrm += Am[0][(t0-1)/5] * Am[0][(t0-1)/5];
            }
          }
          MPI_Barrier_time(MPI_COMM_WORLD);
        }
        if (M <= 2) {
```

```
      for (int ilpp=max(lR,(t0+4)/5);ilpp<=min(uR,(t0+4)/5);ilpp++) {
        if ((t0+4)%5 == 0) {
285       nrm += Am[0][(t0-1)/5] * Am[0][(t0-1)/5];
        }
      }
      MPI_Barrier_time(MPI_COMM_WORLD);
    }
290 if (t0%5 == 0) {
      // Qm[k][(t0-5)/5] in Am[k][ilpp-1] = Am[k][ilpp-1] - Qm[k][(t0-5)/5] *
↪ Rm[(t0-5)/5][ilpp-1]
      for (int k=0;k<=M-1;k++) {
        line_Q_read_vp(0, (t0-5)/5, ceild(t0+7,5), min(floord(t0+5*M-8,5),N), Qm[k], eta_Q_col,
↪ T_Q + (k));
      }
295   //
      for (int ilpp=max(lR,ceild(t0+7,5));ilpp<=min(uR,min(floord(t0+5*M-8,5),N));ilpp++) {
        for (int k=0;k<=M-1;k++) {
          Am[k][ilpp-1] = Am[k][ilpp-1] - Qm[k][(t0-5)/5] * Rm[(t0-5)/5][ilpp-1];
        }
300   }
      MPI_Barrier_time(MPI_COMM_WORLD);
    }
    if ((t0+1)%5 == 0) {
      // Qm[0][(t0-4)/5] in Rm[(t0-4)/5][ilpp-1] += Qm[0][(t0-4)/5] * Am[0][ilpp-1]
305   line_Q_read_vp(0, (t0-4)/5, ceild(t0+7,5), min(floord(t0+5*M-8,5),N), Qm[0], eta_Q_col,
↪ T_Q + 0);
      //
      for (int ilpp=max(lR,ceild(t0+7,5));ilpp<=min(uR,min(floord(t0+5*M-8,5),N));ilpp++) {
        Rm[(t0-4)/5][ilpp-1] += Qm[0][(t0-4)/5] * Am[0][ilpp-1];
      }
310   MPI_Barrier_time(MPI_COMM_WORLD);
    }
    if ((t0+2)%5 == 0) {
      // Am[(-t0+5*ilpp+3)/5][(t0-3)/5] in Qm[(-t0+5*ilpp+3)/5][(t0-3)/5] =
↪ Am[(-t0+5*ilpp+3)/5][(t0-3)/5] / Rm[(t0-3)/5][(t0-3)/5]
      col_Q_read_vp(1, (-t0+3)/5, ceild(t0+7,5), min(floord(t0+5*M-8,5),N), (t0-3)/5, buf_Am,
↪ Am, eta_A, T_A);
315   //
      // Rm[(t0-3)/5][(t0-3)/5] in Qm[(-t0+5*ilpp+3)/5][(t0-3)/5] =
↪ Am[(-t0+5*ilpp+3)/5][(t0-3)/5] / Rm[(t0-3)/5][(t0-3)/5]
      line_Q_read_vp(0, (t0-3)/5, ceild(t0+7,5), min(floord(t0+5*M-8,5),N), Rm[(t0-3)/5],
↪ eta_Q_col, T_R + ((t0-3)/5));
      //
      for (int ilpp=max(lR,ceild(t0+7,5));ilpp<=min(uR,min(floord(t0+5*M-8,5),N));ilpp++) {
320     Qm[(-t0+5*ilpp+3)/5][(t0-3)/5] = Am[(-t0+5*ilpp+3)/5][(t0-3)/5] / Rm[(t0-3)/5][(t0-3)/5];
        Rm[(t0-3)/5][ilpp-1] = 0.0;
      }
      MPI_Barrier_time(MPI_COMM_WORLD);
      // Qm[(-t0+5*ilpp+3)/5][(t0-3)/5] in Qm[(-t0+5*ilpp+3)/5][(t0-3)/5] =
↪ Am[(-t0+5*ilpp+3)/5][(t0-3)/5] / Rm[(t0-3)/5][(t0-3)/5]
325   col_Q_write_vp(1, (-t0+3)/5, ceild(t0+7,5), min(floord(t0+5*M-8,5),N), (t0-3)/5, buf_Qm,
↪ Qm, eta_Q, T_Q);
      //
    }
    if ((t0+2)%5 == 0) {
      // Am[(-t0+5*ilpp+3)/5][(t0-3)/5] in Qm[(-t0+5*ilpp+3)/5][(t0-3)/5] =
↪ Am[(-t0+5*ilpp+3)/5][(t0-3)/5] / Rm[(t0-3)/5][(t0-3)/5]
330   col_Q_read_vp(1, (-t0+3)/5, N+1, floord(t0+5*M-8,5), (t0-3)/5, buf_Am, Am, eta_A, T_A);
      //
      // Rm[(t0-3)/5][(t0-3)/5] in Qm[(-t0+5*ilpp+3)/5][(t0-3)/5] =
↪ Am[(-t0+5*ilpp+3)/5][(t0-3)/5] / Rm[(t0-3)/5][(t0-3)/5]
```

```
      line_Q_read_vp(0, (t0-3)/5, N+1, floord(t0+5*M-8,5), Rm[(t0-3)/5], eta_R_col, T_R +
↪ ((t0-3)/5));
      //
      for (int ilpp=max(lR,N+1);ilpp<=min(uR,floord(t0+5*M-8,5));ilpp++) {
        Qm[(-t0+5*ilpp+3)/5][(t0-3)/5] = Am[(-t0+5*ilpp+3)/5][(t0-3)/5] / Rm[(t0-3)/5][(t0-3)/5];
      }
      MPI_Barrier_time(MPI_COMM_WORLD);
      // Qm[(-t0+5*ilpp+3)/5][(t0-3)/5] in Qm[(-t0+5*ilpp+3)/5][(t0-3)/5] =
↪ Am[(-t0+5*ilpp+3)/5][(t0-3)/5] / Rm[(t0-3)/5][(t0-3)/5]
      col_Q_write_vp(1, (-t0+3)/5, N+1, floord(t0+5*M-8,5), (t0-3)/5, buf_Qm, Qm, eta_Q, T_Q);
      //
    }
    if (t0%5 == 0) {
      // Qm[k][(t0-5)/5] in Am[k][ilpp-1] = Am[k][ilpp-1] - Qm[k][(t0-5)/5] *
↪ Rm[(t0-5)/5][ilpp-1]
      for (int k=0;k<=M-1;k++) {
        line_Q_read_vp(0, (t0-5)/5, max(ceild(t0+7,5),ceild(t0+5*M-7,5)), N, Qm[k], eta_Q_col,
↪ T_Q + (k));
      }
      //
      for (int ilpp=max(lR,max(ceild(t0+7,5),ceild(t0+5*M-7,5)));ilpp<=min(uR,N);ilpp++) {
        for (int k=0;k<=M-1;k++) {
          Am[k][ilpp-1] = Am[k][ilpp-1] - Qm[k][(t0-5)/5] * Rm[(t0-5)/5][ilpp-1];
        }
      }
      MPI_Barrier_time(MPI_COMM_WORLD);
    }
    if ((t0+1)%5 == 0) {
      // Qm[0][(t0-4)/5] in Rm[(t0-4)/5][ilpp-1] += Qm[0][(t0-4)/5] * Am[0][ilpp-1]
      line_Q_read_vp(0, (t0-4)/5, max(ceild(t0+7,5),ceild(t0+5*M-7,5)), N, Qm[0], eta_Q_col, T_Q
↪ + 0);
      //
      for (int ilpp=max(lR,max(ceild(t0+7,5),ceild(t0+5*M-7,5)));ilpp<=min(uR,N);ilpp++) {
        Rm[(t0-4)/5][ilpp-1] += Qm[0][(t0-4)/5] * Am[0][ilpp-1];
      }
      MPI_Barrier_time(MPI_COMM_WORLD);
    }
    if ((t0+2)%5 == 0) {
      for (int ilpp=max(lR,max(ceild(t0+7,5),ceild(t0+5*M-7,5)));ilpp<=min(uR,N);ilpp++) {
        Rm[(t0-3)/5][ilpp-1] = 0.0;
      }
      MPI_Barrier_time(MPI_COMM_WORLD);
    }
    for (int t1=1;t1<=M-1;t1++) {
      for (int ilpp=max(lR,(t0+4)/5);ilpp<=min(uR,(t0+4)/5);ilpp++) {
        if ((t0+4)%5 == 0) {
          nrm += Am[t1][(t0-1)/5] * Am[t1][(t0-1)/5];
        }
      }
      MPI_Barrier_time(MPI_COMM_WORLD);
      if ((t0+1)%5 == 0) {
        // Qm[t1][(t0-4)/5] in Rm[(t0-4)/5][ilpp-1] += Qm[t1][(t0-4)/5] * Am[t1][ilpp-1]
        line_Q_read_vp(0, (t0-4)/5, ceild(t0+6,5), N, Qm[t1], eta_Q_col, T_Q + (t1));
        //
        for (int ilpp=max(lR,ceild(t0+6,5));ilpp<=min(uR,N);ilpp++) {
          Rm[(t0-4)/5][ilpp-1] += Qm[t1][(t0-4)/5] * Am[t1][ilpp-1];
        }
        MPI_Barrier_time(MPI_COMM_WORLD);
      }
    }
  }
```

```
  }
  if ((M >= 1) && (N >= 3)) {
    for (int t1=0;t1<=M-1;t1++) {
      // Qm[t1][N-2] in Rm[N-2][N-1] += Qm[t1][N-2] * Am[t1][N-1]
      line_Q_read_vp(0, N-2, N, N, Qm[t1], eta_Q_col, T_Q + (t1));
      //
      for (int ilpp=max(lR,N);ilpp<=min(uR,N);ilpp++) {
        Rm[N-2][N-1] += Qm[t1][N-2] * Am[t1][N-1];
      }
      MPI_Barrier_time(MPI_COMM_WORLD);
    }
  }
  if ((M >= 1) && (N >= 2)) {
    for (int ilpp=max(lR,N-1);ilpp<=min(uR,N-1);ilpp++) {
      nrm = 0.0;
    }
    MPI_Barrier_time(MPI_COMM_WORLD);
    // Qm[k][N-2] in Am[k][N-1] = Am[k][N-1] - Qm[k][N-2] * Rm[N-2][N-1]
    for (int k=0;k<=M-1;k++) {
      line_Q_read_vp(0, N-2, N, N, Qm[k], eta_Q_col, T_Q + (k));
    }
    //
    for (int ilpp=max(lR,N);ilpp<=min(uR,N);ilpp++) {
      for (int k=0;k<=M-1;k++) {
        Am[k][N-1] = Am[k][N-1] - Qm[k][N-2] * Rm[N-2][N-1];
      }
    }
    MPI_Barrier_time(MPI_COMM_WORLD);
  }
  if ((M >= 1) && (N >= 2)) {
    for (int t1=0;t1<=M-1;t1++) {
      for (int ilpp=max(lR,N);ilpp<=min(uR,N);ilpp++) {
        nrm += Am[t1][N-1] * Am[t1][N-1];
      }
      MPI_Barrier_time(MPI_COMM_WORLD);
    }
  }
  if ((M >= 1) && (N >= 2)) {
    for (int ilpp=max(lR,N);ilpp<=min(uR,N);ilpp++) {
      Rm[N-1][N-1] = sqrt(nrm);
    }
    MPI_Barrier_time(MPI_COMM_WORLD);
  }
  if (M >= 1) {
    // Am[ilpp-N+1][N-1] in Qm[ilpp-N+1][N-1] = Am[ilpp-N+1][N-1] / Rm[N-1][N-1]
    col_Q_read_vp(1, -N+1, N-1, N+M-2, N-1, buf_Am, Am, eta_A, T_A);
    //
    // Rm[N-1][N-1] in Qm[ilpp-N+1][N-1] = Am[ilpp-N+1][N-1] / Rm[N-1][N-1]
    line_Q_read_vp(0, N-1, N-1, N+M-2, Rm[N-1], eta_R_col, T_R + (N-1));
    //
    for (int ilpp=max(lR,N-1);ilpp<=min(uR,N+M-2);ilpp++) {
      Qm[ilpp-N+1][N-1] = Am[ilpp-N+1][N-1] / Rm[N-1][N-1];
    }
    MPI_Barrier_time(MPI_COMM_WORLD);
    // Qm[ilpp-N+1][N-1] in Qm[ilpp-N+1][N-1] = Am[ilpp-N+1][N-1] / Rm[N-1][N-1]
    col_Q_write_vp(1, -N+1, N-1, N+M-2, N-1, buf_Qm, Qm, eta_Q, T_Q);
    //
  }
  collect_matrix_cols_double_offset(Am, M, N, A_chunk_size, eta_A_col(0));
  collect_matrix_cols_double_offset(Qm, M, N, Q_chunk_size, eta_Q_col(0));
  collect_matrix_cols_double_offset(Rm, N, N, R_chunk_size, eta_R_col(0));
```

```
  }
}
```

E.4 Преобразования pluto

Листинг E.8 Преобразования pluto в формате cloog для gramschmidt_pluto

```
# Number of scattering functions
7

# T(S1)
6 11
0 1 0 0 0 0 0 0 0 0 -1
0 0 1 0 0 0 0 -1 0 0 0
0 0 0 1 0 0 0 0 0 0 -1
0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 -1
0 0 0 0 0 0 1 0 0 0 0

# T(S2)
6 12
0 1 0 0 0 0 0 0 0 0 0 -1
0 0 1 0 0 0 0 -1 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 -2
0 0 0 0 1 0 0 0 -1 0 0 0
0 0 0 0 0 1 0 0 0 0 0 -1
0 0 0 0 0 0 1 0 0 0 0 0

# T(S3)
6 11
0 1 0 0 0 0 0 0 0 0 -1
0 0 1 0 0 0 0 -1 0 0 0
0 0 0 1 0 0 0 0 0 0 -3
0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 -1
0 0 0 0 0 0 1 0 0 0 0

# T(S4)
6 12
0 1 0 0 0 0 0 0 0 0 0 -1
0 0 1 0 0 0 0 -1 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 -4
0 0 0 0 1 0 0 0 -1 0 0 0
0 0 0 0 0 1 0 0 0 0 0 -1
0 0 0 0 0 0 1 0 0 0 0 0

# T(S5)
6 12
0 1 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 -1 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 -1 0 0 0
```

```
0 0 0 0 0 1 0 0 0 0 0 -1
0 0 0 0 0 0 1 0 0 0 0 0

# T(S6)
6 13
0 1 0 0 0 0 0 0 0 0 0 0 -1
0 0 1 0 0 0 0 -1 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 -5
0 0 0 0 1 0 0 0 -1 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 -1 0 0 0

# T(S7)
6 13
0 1 0 0 0 0 0 0 0 0 0 0 -1
0 0 1 0 0 0 0 -1 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 -5
0 0 0 0 1 0 0 0 -1 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 -1
0 0 0 0 0 0 1 0 0 -1 0 0 0
```

Листинг E.9 gramschmidt (параллельный вариант pluto с директивами OpenMP, синхронный параллелизм) на языке C

```
 int t1, t2, t3, t4, t5, t6;
 int lb, ub, lbd, ubd, lb2, ub2;
 register int lbv, ubv;
/* Start of CLooG code */
if (N >= 1) {
 {/* extra braces to avoid redefination of lbp*/
int lbp=0;
 int ubp=N-2;
#pragma omp parallel for private(lbv,ubv,t3,t4,t5,t6)
 for (t2=lbp;t2<=ubp;t2++) {
  lbv=t2+1;
  ubv=N-1;
#pragma ivdep
#pragma vector always
  for (t4=lbv;t4<=ubv;t4++) {
   R[t2][t4] = 0.0;;
  }
 }
}/*end of omp parallel loop */
 if (M >= 1) {
  for (t2=0;t2<=N-2;t2++) {
   nrm = 0.0;;
   for (t4=0;t4<=M-1;t4++) {
    nrm += A[t4][t2] * A[t4][t2];;
   }
   R[t2][t2] = nrm;;
   {/* extra braces to avoid redefination of lbp*/
int lbp=0;
   int ubp=M-1;
#pragma omp parallel for private(lbv,ubv,t5,t6)
   for (t4=lbp;t4<=ubp;t4++) {
    Q[t4][t2] = A[t4][t2] / R[t2][t2];;
   }
}/*end of omp parallel loop */
   {/* extra braces to avoid redefination of lbp*/
int lbp=t2+1;
```

```
   int ubp=N-1;
#pragma omp parallel for private(lbv,ubv,t5,t6)
   for (t4=lbp;t4<=ubp;t4++) {
    for (t6=0;t6<=M-1;t6++) {
     R[t2][t4] += Q[t6][t2] * A[t6][t4];;
    }
    lbv=0;
    ubv=M-1;
#pragma ivdep
#pragma vector always
    for (t6=lbv;t6<=ubv;t6++) {
     A[t6][t4] = A[t6][t4] - Q[t6][t2] * R[t2][t4];;
    }
   }
}/*end of omp parallel loop */
  }
 }
 if (M >= 1) {
  nrm = 0.0;;
  for (t4=0;t4<=M-1;t4++) {
   nrm += A[t4][(N-1)] * A[t4][(N-1)];;
  }
  R[(N-1)][(N-1)] = nrm;;
  {/* extra braces to avoid redefination of lbp*/
int lbp=0;
  int ubp=M-1;
#pragma omp parallel for private(lbv,ubv,t5,t6)
  for (t4=lbp;t4<=ubp;t4++) {
   Q[t4][(N-1)] = A[t4][(N-1)] / R[(N-1)][(N-1)];;
  }
}/*end of omp parallel loop */
 }
 if (M <= 0) {
  for (t2=0;t2<=N-1;t2++) {
   nrm = 0.0;;
   R[t2][t2] = nrm;;
  }
 }
}
/* End of CLooG code */
```

Листинг E.10 gramschmidt_pluto (обработанный вариант pluto с директивами OpenMP, синхронный параллелизм) на языке C

```
void gramschmidt_pluto(int M, int N, double** A, double** Q, double** R, double* nrm) {
  #pragma omp parallel
  {
    /* Start of CLooG code */
    if (N >= 1) {
      #pragma omp for
      for (int t2=0;t2<=N-2;t2++) {
        for (int t4=t2+1;t4<=N-1;t4++) {
          R[t2][t4] = 0.0;
        }
      }
      if (M >= 1) {
        for (int t2=0;t2<=N-2;t2++) {
          #pragma omp single
          {
            *nrm = 0.0;
```

```
            for (int t4=0;t4<=M-1;t4++) {
              *nrm += A[t4][t2] * A[t4][t2];
            }
            R[t2][t2] = sqrt(*nrm);
          }
          #pragma omp for
          for (int t4=0;t4<=M-1;t4++) {
            Q[t4][t2] = A[t4][t2] / R[t2][t2];
          }
          #pragma omp for
          for (int t4=t2+1;t4<=N-1;t4++) {
            for (int t6=0;t6<=M-1;t6++) {
              R[t2][t4] += Q[t6][t2] * A[t6][t4];
            }
            for (int t6=0;t6<=M-1;t6++) {
              A[t6][t4] = A[t6][t4] - Q[t6][t2] * R[t2][t4];
            }
          }
        }
      }
      if (M >= 1) {
        #pragma omp single
        {
          *nrm = 0.0;
          for (int t4=0;t4<=M-1;t4++) {
            *nrm += A[t4][(N-1)] * A[t4][(N-1)];
          }
          R[(N-1)][(N-1)] = sqrt(*nrm);
        }
        #pragma omp for
        for (int t4=0;t4<=M-1;t4++) {
          Q[t4][(N-1)] = A[t4][(N-1)] / R[(N-1)][(N-1)];
        }
      }
      if (M <= 0) {
        #pragma omp single
        for (int t2=0;t2<=N-1;t2++) {
          *nrm = 0.0;
          R[t2][t2] = sqrt(*nrm);
        }
      }
    }
    /* End of CLooG code */
  }
}
```

E.5 Статистика запусков gramschmidt (OpenMP)

Таблица 39 — Статистика запусков gramschmidt в вариантах vanilla, ilp_sync, pluto: затраченное время, мс

#Нитей	Стат.	vanilla	ilp_sync	pluto
1	μ	41263.7153	17811.5612	41097.6674
	σ	61.502301	50.584789	144.666563
	Min	41219.388	17787.239	40999.191
	Max	41441.405	17961.268	41457.769
2	μ	–	12146.7367	21317.0137
	σ	–	28.81905	106.317367
	Min	–	12116.298	21161.342
	Max	–	12196.712	21458.352
4	μ	–	8089.3131	11153.411
	σ	–	0.914839	0.712743
	Min	–	8087.53	11152.83
	Max	–	8090.755	11155.299
6	μ	–	6281.7513	7554.4961
	σ	–	1.012799	9.831074
	Min	–	6280.301	7542.546
	Max	–	6283.999	7571.662
8	μ	–	5381.9292	5671.0888
	σ	–	9.859556	14.596475
	Min	–	5374.483	5650.894
	Max	–	5397.306	5689.571

E.6 Статистика запусков gramschmidt (MPI)

Таблица 40 — Статистика запусков gramschmidt_mpi в вариантах vanilla, ilp_arrays (_one, _two, _eq): затраченное время, мс

#Проц.	Стат.	vanilla	ilp_arrays_one	ilp_arrays_two	ilp_arrays_eq
1	μ	40524.24764	22348.52593	–	–
	σ	24.301836	15.508634	–	–
	Min	40478.802	22334.838	–	–
	Max	40617.531	22461.867	–	–
2	μ	–	27008.00306	47737.10444	–
	σ	–	16.792056	225.007648	–
	Min	–	26991.218	47400.051	–
	Max	–	27088.211	48103.079	–
4	μ	–	22798.84425	44348.94891	81619.44481
	σ	–	4.977218	212.592729	1043.042557
	Min	–	22789.953	44091.425	79567.085
	Max	–	22827.487	44721.888	84114.598

Продолжение таблицы 40

#Проц.	Стат.	vanilla	ilp_arrays_one	ilp_arrays_two	ilp_arrays_eq
6	μ	–	25583.27002	48030.81327	111447.36929
	σ	–	60.047579	1165.162682	5704.789672
	Min	–	25497.444	47040.929	103338.547
	Max	–	25659.876	50136.016	118395.603
8	μ	–	21393.92518	44316.92374	95161.1406
	σ	–	19.707186	332.618201	188.585259
	Min	–	21355.857	43864.496	94686.65
	Max	–	21433.762	45053.182	95627.455

Таблица 41 — Статистика запусков gramschmidt_mpi в вариантах vanilla, ilp_arrays (_one, _two, _eq): доля времени вычислений, %

#Проц.	Стат.	vanilla	ilp_arrays_one	ilp_arrays_two	ilp_arrays_eq
1	μ	100.0	97.716309	–	–
	σ	0.0	0.003689	–	–
	Min	100.0	97.708609	–	–
	Max	100.0	97.725051	–	–
2	μ	–	42.796653	23.700178	–
	σ	–	0.004062	0.111109	–
	Min	–	42.789006	23.525835	–
	Max	–	42.806425	23.847896	–
4	μ	–	27.993329	13.604484	7.60179
	σ	–	0.005282	0.065467	0.082218
	Min	–	27.979935	13.503479	7.407845
	Max	–	28.005415	13.683924	7.748424
6	μ	–	17.367248	9.209745	3.769861
	σ	–	0.041743	0.062092	0.097644
	Min	–	17.320759	9.076802	3.629185
	Max	–	17.426871	9.291376	3.931367
8	μ	–	16.153593	7.848439	3.331211
	σ	–	0.015084	0.037895	0.007569
	Min	–	16.124518	7.786152	3.313076
	Max	–	16.182192	7.905254	3.346484

Таблица 42 — Разнородная нагрузка в запусках gramschmidt_mpi

Запуск	Затраченное время, мс			Доля затраченного времени, %		
vanilla: 1=1x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	40523.44	0.00	0.00	100.00	0.00	0.00
ilp_arrays_one: 1=1x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	22346.38	213.54	296.99	97.72	0.96	1.33
ilp_arrays_one: 2=1x2	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	27006.24	1895.79	2495.15	83.74	7.02	9.24
Process 1	27006.24	1894.64	24610.35	1.86	7.02	91.13
ilp_arrays_two: 2=2x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	47816.39	10167.58	15611.85	46.09	21.26	32.65
Process 1	47816.35	10797.71	36516.27	1.05	22.58	76.37
ilp_arrays_one: 4=1x4	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	22798.28	2944.63	6154.36	60.09	12.92	26.99
Process 1	22798.28	2946.76	9056.32	47.35	12.93	39.72
Process 2	22798.28	2941.45	19335.52	2.29	12.90	84.81
Process 3	22798.28	2942.44	19338.64	2.27	12.91	84.82
ilp_arrays_two: 4=2x2	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	44507.44	11816.59	19837.99	28.88	26.55	44.57

Продолжение таблицы 42

Запуск	Затраченное время, мс			Доля затраченного времени, %		
Process 1	44507.44	11823.62	22410.96	23.08	26.57	50.35
Process 2	44507.44	12271.09	31737.57	1.12	27.57	71.31
Process 3	44507.44	12268.93	31742.44	1.11	27.57	71.32
ilp_arrays_eq: 4=4x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	81632.79	31442.24	36917.42	16.26	38.52	45.22
Process 1	81632.76	31602.25	39535.05	12.86	38.71	48.43
Process 2	81632.76	31994.33	49109.53	0.65	39.19	60.16
Process 3	81632.79	31690.15	49423.11	0.64	38.82	60.54
ilp_arrays_one: 6=1x6	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	25620.10	3632.52	15630.17	24.81	14.18	61.01
Process 1	25620.10	3638.66	3851.58	70.76	14.20	15.03
Process 2	25620.10	3638.15	21361.76	2.42	14.20	83.38
Process 3	25620.10	3636.91	21463.57	2.03	14.20	83.78
Process 4	25620.10	3634.22	21472.30	2.00	14.19	83.81
Process 5	25620.10	3635.84	21472.52	2.00	14.19	83.81
ilp_arrays_two: 6=2x3	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	47543.22	12283.90	29072.33	13.01	25.84	61.15
Process 1	47543.22	12282.76	17328.73	37.72	25.83	36.45
Process 2	47543.22	12283.55	34650.18	1.28	25.84	72.88
Process 3	47543.23	12847.28	34186.43	1.07	27.02	71.91
Process 4	47543.23	12850.06	34184.14	1.07	27.03	71.90
Process 5	47543.23	12849.31	34188.64	1.06	27.03	71.91
ilp_arrays_eq: 6=6x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	112153.23	44308.46	61778.75	5.41	39.51	55.08
Process 1	112153.25	44432.38	50506.59	15.35	39.62	45.03
Process 2	112153.23	44240.59	67298.36	0.55	39.45	60.01
Process 3	112153.25	43889.39	67749.32	0.46	39.13	60.41
Process 4	112153.23	44321.15	67317.95	0.46	39.52	60.02
Process 5	112153.22	44441.90	67225.03	0.43	39.63	59.94
ilp_arrays_one: 8=1x8	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	21393.85	3816.25	13790.59	17.70	17.84	64.46
Process 1	21393.85	3807.29	7141.90	48.82	17.80	33.38
Process 2	21393.85	3805.60	6758.57	50.62	17.79	31.59
Process 3	21393.85	3810.94	17066.41	2.41	17.81	79.77
Process 4	21393.85	3811.38	17062.74	2.43	17.82	79.76
Process 5	21393.85	3813.60	17063.96	2.41	17.83	79.76
Process 6	21393.85	3811.19	17064.89	2.42	17.81	79.77
Process 7	21393.85	3816.36	17064.48	2.40	17.84	79.76
ilp_arrays_two: 8=2x4	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	44313.59	12786.42	27760.55	8.50	28.85	62.65
Process 1	44313.59	12783.70	21082.02	23.58	28.85	47.57
Process 2	44313.59	12783.08	20672.96	24.50	28.85	46.65
Process 3	44313.59	12785.24	31002.48	1.19	28.85	69.96
Process 4	44313.54	13193.44	30592.64	1.19	29.77	69.04
Process 5	44313.54	13199.34	30594.12	1.17	29.79	69.04
Process 6	44313.54	13192.74	30598.60	1.18	29.77	69.05
Process 7	44313.54	13196.56	30595.80	1.18	29.78	69.04
ilp_arrays_eq: 8=8x1	Исполнение	Обмен	Синхронизация	Вычисления	Обмен	Синхронизация
Process 0	95162.46	38006.09	53654.91	3.68	39.94	56.38
Process 1	95162.46	38200.22	47463.62	9.98	40.14	49.88
Process 2	95162.48	38215.61	47058.93	10.39	40.16	49.45
Process 3	95162.46	37981.04	56692.19	0.51	39.91	59.57
Process 4	95162.47	38380.25	56291.50	0.52	40.33	59.15
Process 5	95162.46	38714.88	55956.84	0.52	40.68	58.80
Process 6	95162.45	38537.28	56141.52	0.51	40.50	59.00
Process 7	95162.48	38319.68	56359.81	0.51	40.27	59.22

Приложение Ж

Специальные возможности ilpy: оценка аффинных отображений инструкций

При нахождении расписаний вычислений, а также размещений вычислений и данных, ilpy вычисляет значение целевых функций (2.5) и (2.15). При необходимости сравнить аффинные отображения, вычисленные ilpy, с афинными отображениями, вычисленными применением других подходов и средств, встает задача оценки рассматриваемых решений согласно введенным в главе 2 критериям качества. Рассмотрим оценку аффинных отображений инструкций с помощью утверждения 1. Она полагается на следующую систематическую процедуру нахождения для каждого ребра e в обобщенном графе зависимостей верхней границы $L_e(\vec{z}) = \vec{l_e} \cdot \vec{z} + l_e^0$: $-|\varphi_{\delta(e)}(\vec{i}_{\delta(e)}, \vec{z}) - \varphi_{\sigma(e)}(\vec{i}_{\sigma(e)}, \vec{z})| + L_e(\vec{z}) \geqslant 0$.

1. Построить систему ограничений:

$$\begin{cases} -\varphi_{\delta(e)}(\vec{i}_{\delta(e)}, \vec{z}) + \varphi_{\sigma(e)}(\vec{i}_{\sigma(e)}, \vec{z}) + \vec{l_e} \cdot \vec{z} + l_e^0 \geqslant 0 \\ \varphi_{\delta(e)}(\vec{i}_{\delta(e)}, \vec{z}) - \varphi_{\sigma(e)}(\vec{i}_{\sigma(e)}, \vec{z}) + \vec{l_e} \cdot \vec{z} + l_e^0 \geqslant 0 \end{cases} . \qquad \text{(Ж.1)}$$

2. Решить задачу

$$\vec{l_e} \cdot \vec{z} + l_e^0 \to \min \qquad \text{(Ж.2)}$$

при ограничениях (Ж.1). Задача решается методами линейного целочисленного программирования после линеаризации системы ограничений (классическая техника с применением леммы Фаркаша).

Взвешенная сумма вычисленных $L_e(\vec{z})$ для всех ребер e в обобщенном графе зависимостей дает значение целевой функции (2.5).

Первое условие в системе (Ж.1) имеет вид `DEP_UPPER_BOUND`, а второе — вид `DEP_LOWER_BOUND`. Каждое из них выполняется на многограннике зависимостей R_e, что позволяет применить лемму Фаркаша, а затем метод неопределенных коэффициентов. Линеаризация системы ограничений упрощается, поскольку для известных аффинных отображений не выписывается шаблон с применением леммы Фаркаша. Приравнивание коэффициентов при индексах итерации, индексах массива, внешних переменных программы, а также констант, порождает новые условия в виде равенств для рассматриваемой зависимости по данным e. Формируются 6 групп ограничений в виде равенств. Новые ограничения представляются в виде строк матрицы, имеющей структуру, представленную в таблице 43.

Таблица 43 — Структура матрицы для хранения ограничений в виде равенств

		$\lambda_{e,k}$, $k = 1, \ldots, p_{R_e}$	$\lambda_{e,0}$	$\lambda'_{e,k}$, $k = 1, \ldots, p_{R_e}$	$\lambda'_{e,0}$	$\vec{l}_e^{(k)}$, $k = 1, \ldots, q_z$	l_e^0
Для каждого условия	$\vec{i}^{(v)}_{\sigma(e_j)}$, $v = 1, \ldots, p_{\sigma(e_j)}$						
	$\vec{g}^{(v)}_{a_{\sigma(e_j)}}$, $v = 1, \ldots, p_A + 1$						
	$\vec{i}^{(v)}_{\delta(e_j)}$, $v = 1, \ldots, p_{\delta(e_j)}$						
	$\vec{g}^{(v)}_{a_{\delta(e_j)}}$, $v = 1, \ldots, p_A + 1$						
	$\vec{z}^{(v)}$, $v = 1, \ldots, q_z$						
	1						

Значение элемента матрицы (i,j) определяется коэффициентом при переменной, указанной в названии столбца j, в равенстве, определяемом строкой i (с учетом множителей μ_S^{sign}). Если переменная в равенстве не участвует, то значение соответствующего элемента матрицы считается нулевым, но в память не заносится, так как матрица хранится в формате списка координат, ориентированном на хранение разреженных матриц и поддерживаемом библиотекой glpk [115].

1. Для $\vec{i}^{(v)}_{\sigma(e)}, \quad v = 1, \ldots, p_{\sigma(e)}$:

$$- \sum_{k=source^{start}_{domain}}^{source^{end}_{domain}} \lambda^*_{e,k+1} m_{R_e}[k][v-1] - \sum_{k=source^{start}_{access}}^{source^{end}_{access}} \lambda^*_{e,k+1} m_{R_e}[k][v-1] - \\ - \sum_{k=precedence^{start}}^{precedence^{end}} \lambda^*_{e,k+1} m_{R_e}[k][v-1] = -\mu^{sign}_{\sigma(e)} \vec{v}^{(v)}_{\sigma(e)}.$$

2. Для $\vec{g}^{(v)}_{a_{\sigma(e)}}, \quad v = 1, \ldots, p_A + 1$:

$$- \sum_{k=source^{start}_{access}}^{source^{end}_{access}} \lambda^*_{e,k+1} m_{R_e}[k][v-1+p_{\sigma(e)}] - \\ - \sum_{k=access^{start}_{eq}}^{access^{end}_{eq}} \lambda^*_{e,k+1} m_{R_e}[k][v-1+p_{\sigma(e)}] = 0.$$

3. Для $\vec{i}^{(v)}_{\delta(e)}, \quad v = 1, \ldots, p_{\delta(e)}$:

$$- \sum_{k=target^{start}_{domain}}^{target^{end}_{domain}} \lambda^*_{e,k+1} m_{R_e}[k][v - 1 + p_{\sigma(e)} + (p_A + 1)] -$$

$$- \sum_{k=target^{start}_{access}}^{target^{end}_{access}} \lambda^*_{e,k+1} m_{R_e}[k][v - 1 + p_{\sigma(e)} + (p_A + 1)] -$$

$$- \sum_{k=precedence^{start}}^{precedence^{end}} \lambda^*_{e,k+1} m_{R_e}[k][v - 1 + p_{\sigma(e)} + (p_A + 1)] = -\mu^{sign}_{\delta(e)} \vec{v}^{(v)}_{\delta(e)}.$$

4. Для $\vec{g}^{(v)}_{a_{\delta(e)}}, \quad v = 1, \ldots, p_A + 1$:

$$- \sum_{k=target^{start}_{access}}^{target^{end}_{access}} \lambda^*_{e,k+1} m_{R_e}[k][v - 1 + p_{\sigma(e)} + (p_A + 1) + p_{\delta(e)}] -$$

$$- \sum_{k=access^{start}_{eq}}^{access^{end}_{eq}} \lambda^*_{e,k+1} m_{R_e}[k][v - 1 + p_{\sigma(e)} + (p_A + 1) + p_{\delta(e)}] = 0.$$

5. Для $\vec{z}^{(v)}, \quad v = 1, \ldots, q_z$:

$$- \sum_{k=source^{start}_{domain}}^{target^{end}_{access}} \lambda^*_{e,k+1} m_{R_e}[k][v - 1 + p_{\sigma(e)} + 2(p_A + 1) + p_{\delta(e)}] + \vec{l}^{(v)}_e =$$

$$= -\mu^{sign}_{\sigma(e)} \vec{v}'^{(v)}_{\sigma(e)} - \mu^{sign}_{\delta(e)} \vec{v}'^{(v)}_{\delta(e)}.$$

6. Для констант:

$$-\lambda^*_{e,0} - \sum_{k=source^{start}_{domain}}^{target^{end}_{access}} \lambda^*_{e,k+1} m_{R_e}[k][p_{\sigma(e)} + 2(p_A + 1) + p_{\delta(e)} + q_z] -$$

$$- \sum_{k=precedence^{start}}^{precedence^{end}} \lambda^*_{e,k+1} m_{R_e}[k][p_{\sigma(e)} + 2(p_A + 1) + p_{\delta(e)} + q_z] + l^0_e =$$

$$= -\mu^{sign}_{\sigma(e)} v^0_{\sigma(e)} - \mu^{sign}_{\delta(e)} v^0_{\delta(e)}.$$

Для первого условия в системе (Ж.1) λ^* соответствует λ', для второго — λ. Настройка решателя glpk [115] совпадает с таковой для поиска компонента многомерного расписания вычислений.

В таблице 44 представлена статистика для всех значений целевой функции (2.5), соответствующих компонентам многомерного аффинного отображения Φ, найденного транслятором. Компоненты, являющиеся константами для всех инструкций, не рассматривались: размерность $\dim \Phi$ их также не учитывает. Для каждого из столбцов (сумма $\sum C$, математическое ожидание μ_C, стандартное отклонение σ_C, минимальное значение $\min C$, максимальное значение $\max C$, размерность отображения $\dim \Phi$) вычислен коэффициент корреляции Пирсона относительно коэффициента ускорения вычислений (строка r).

Таблица 44 — Статистика для целевой функции (2.5)

Файл	Процедура	$\sum C$	μ_C	σ_C	$\min C$	$\max C$	$\dim(\Phi)$	Ускорение, раз
lu.c-deps.ilpy	lu_ilp_sync	3.59e+07	1.79e+07	1.44e+07	3.53e+06	3.23e+07	2	2.27
lu.pluto.cloog	lu_pluto	1.31e+08	4.36e+07	1.50e+07	3.23e+07	6.47e+07	3	1.82
atax.c-deps.ilpy	atax_ilp_sync	7.24e+08	2.41e+08	1.98e+08	1.81e+07	5.00e+08	3	1.31
atax.pluto.cloog	atax_pluto	5.14e+08	1.03e+08	2.01e+08	2.00e+04	5.04e+08	5	1.43
atax_p.c-deps.ilpy	atax_p_ilp_sync	2.08e+08	1.04e+08	1.02e+06	1.03e+08	1.05e+08	2	0.92
atax_p.pluto.cloog	atax_p_pluto	2.10e+08	7.00e+07	4.81e+07	2.07e+06	1.05e+08	3	1.13
atax_p.c-deps_mdp.ilpy	atax_p_ilp_sync_mdp	3.09e+08	1.03e+08	8.33e+07	1.08e+06	2.05e+08	3	2.96
syr2k.c-adeps.ilpy	syr2k_ilp_async	2.03e+08	6.78e+07	9.59e+07	0.00e+00	2.03e+08	3	45.88
syr2k.pluto.cloog	syr2k_pluto	2.06e+08	5.14e+07	8.78e+07	0.00e+00	2.03e+08	4	44.73
floyd.c-deps.ilpy	floyd_ilp_sync	6.01e+08	2.00e+08	1.59e+08	6.93e+06	3.96e+08	3	0.64
floyd.pluto.cloog	floyd_pluto	6.01e+08	2.00e+08	1.59e+08	6.93e+06	3.96e+08	3	0.64
gramschmidt.c-deps.ilpy	gramschmidt_ilp_sync	2.57e+09	8.56e+08	5.66e+08	8.21e+07	1.42e+09	3	7.67
gramschmidt.pluto.cloog	gramschmidt_pluto	1.65e+09	2.75e+08	3.97e+08	1.50e+06	1.06e+09	6	7.28
r		-0.13	-0.14	-0.06	-0.23	-0.09	0.14	-

Значения коэффициента корреляции свидетельствуют об отсутствии явной линейной зависимости коэффициента ускорения от какой-либо из рассмотренных в таблице величин.

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2022667001

Программа для построения аффинных преобразований программ линейного класса

Правообладатель: ***Федеральное государственное бюджетное учреждение науки Центр информационных технологий в проектировании Российской академии наук (RU)***

Авторы: ***Лебедев Артем Сергеевич (RU), Солодовников Владимир Игоревич (RU)***

Заявка № **2022666032**
Дата поступления **29 августа 2022 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***13 сентября 2022 г.***

Руководитель Федеральной службы
по интеллектуальной собственности

ДОКУМЕНТ ПОДПИСАН ЭЛЕКТРОННОЙ ПОДПИСЬЮ
Сертификат 68b80077e14e40f0a94edbd24145d5c7
Владелец **Зубов Юрий Сергеевич**
Действителен с 2.03.2022 по 26.05.2023

Ю.С. Зубов

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2016618301

Модуль вычисления пространственно-временного преобразования линейной программы с целью ее распараллеливания и оптимизации локальности данных

Правообладатель: ***Общество с ограниченной ответственностью "Технологии высокопроизводительных вычислений" (RU)***

Автор: ***Лебедев Артем Сергеевич (RU)***

Заявка № **2016612892**
Дата поступления **30 марта 2016 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***26 июля 2016 г.***

Руководитель Федеральной службы
по интеллектуальной собственности

Г.П. Ивлиев

ООО "НПП САТЭК плюс"
www.nppsatek.ru
152934, г. Рыбинск, Ярославская обл., ул. Пушкина, 53
152934, г. Рыбинск, Ярославская обл., ул. Бульварная, 2
ИНН 7610084692, КПП 761001001, р/с 40702810677030008200
в банке ПАО СБЕРБАНК,
к/с 30101810100000000612, БИК 042908612

Исх. от 01.12.2023 № 34

АКТ

о внедрении результатов диссертационной работы

Лебедева Артема Сергеевича

по специальности 2.3.5. «Математическое и программное обеспечение вычислительных систем, комплексов и компьютерных сетей»

«Методы и средства распараллеливания программ линейного класса для выполнения на многопроцессорных вычислительных системах»

Настоящим удостоверяется, что в производственных процессах ООО «НПП САТЭК плюс» при разработке распределенной системы мониторинга и анализа данных о техническом состоянии электронного оборудования нашли применение и внедрены следующие результаты диссертационного исследования Лебедева А.С.:

1. Метод нахождения оптимальных пространственных и временных отображений программ линейного класса для распараллеливания гнезд циклов, отличающийся применением взвешенной суммы показателей качества решения.
2. Метод генерации параллельной MPI-программы, позволяющий организовать информационный обмен между параллельными процессами в случае явно заданного распределения данных между процессорами.

Настоящий акт не является основанием для взаимных финансовых расчетов.

Генеральный директор Петров А.В.

М.П. (печать организации)

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА
просп. Вернадского, д. 78, Москва, 119454
тел.: (499) 215 65 65 доб. 1140, факс: (495) 434 92 87
e-mail: mirea@mirea.ru, http://www.mirea.ru

Утверждаю
Проректор
О.Е. Винокуров
«__» 26 октября 2023

АКТ
о внедрении результатов диссертации на соискание ученой степени кандидата технических наук Лебедева Артема Сергеевича

Настоящим актом подтверждается, что основные результаты диссертационного исследования, выполненного Лебедевым Артемом Сергеевичем, на тему «Методы и средства распараллеливания программ линейного класса для выполнения на многопроцессорных вычислительных системах» внедрены в учебный процесс кафедры КБ-4 «Интеллектуальные системы информационной безопасности» при изучении дисциплины «Методы параллельного программирования», читаемой студентам по направлению подготовки (специальности) 10.05.05 «Безопасность информационных технологий в правоохранительной сфере» и кафедры КБ-14 «Цифровые технологии обработки данных» при изучении дисциплины «Алгоритмы параллельных вычислений», читаемой студентам по направлению подготовки 09.03.02 «Информационные системы и технологии».

И.о. директора Института кибербезопасности и цифровых технологий РТУ МИРЭА
к.ю.н., д.и.н., доцент — А.А. Бакаев

Заведующий кафедрой КБ-4,
к.т.н., доцент — Ш.Г. Магомедов

Заведующий кафедрой КБ-14,
к.т.н., доцент — И.А. Иванова