Пространственно-временные преобразования при распараллеливании линейных программ[1]

А.С. Лебедев

Аннотация. Рассматриваются задачи поиска аффинных расписаний и размещений для линейных программ как задачи многокритериального выбора в условиях полной определенности с целью оптимизации локальности данных. Разработан метод, использующий аппарат модели многогранников и позволяющий минимизировать задержку и расстояние использования данных, исходя из предпочтений лица, принимающего решение (ЛПР). Предпочтения ЛПР задаются набором весовых коэффициентов линейной свертки критериев качества альтернативы. Поиск оптимальной по Парето альтернативы сводится к задаче линейного целочисленного программирования. Разработанный метод позволяет точнее осуществлять задание предпочтений ЛПР (в особенности для программ со слабой или ослабленной параметризацией при Just-In-Time компиляции) по сравнению с распространенным на момент исследования подходом, реализованным в компиляторе Pluto. Применение метода иллюстрируется примером распараллеливания алгоритма LU-разложения. Параллельный вариант алгоритма, полученный с помощью разработанного метода, показывает лучшее быстродействие по сравнению с результатом работы компилятора Pluto.

Ключевые слова: автоматическое распараллеливание, локальность данных, модель многогранников, линейное целочисленное программирование.

Введение

С увеличением количества вычислительных элементов в современных процессорах вычислительные системы приобретают все большую степень параллелизма. Разработка вычислительного программного кода, способного эффективно задействовать возможности аппаратуры, всегда являлась сложной задачей. Сегодня одним из актуальных подходов решения данной задачи является автоматическое распараллеливание программ.

Большинство вычислительно емких научных и инженерных приложений тратят значительную часть времени исполнения на вложенности циклов, часто отвечающие критериям линейности программ. Модель многогранников [1] предоставляет мощный математический аппарат, упрощающий анализ и преобразование таких программ с целью улучшения быстродействия путем повышения степени параллелизма и оптимизации локальности данных при вычислениях.

Методы модели многогранников широко используются в статической компиляции [2-4]. Код, написанный на языке высокого уровня (чаще всего это C или FORTRAN), оптимизируется для параллельного выполнения и компилируется для целевой архитектуры.

Распараллеливание в модели многогранников осуществляется в пять этапов.

1. Анализ информационных зависимостей [5]. Рассматриваются потоковые зависимости (чтение после записи), антизависимости (запись после чтения), зависимости через выход (запись после записи).

[1]Работа выполнена при финансовой поддержке РФФИ проект № 15-07-00925 А и программы фундаментальных исследований ОНИТ 1 РАН «Интеллектуальные информационные технологии, системный анализ и автоматизация».

2. Поиск аффинного расписания программы [4, 6, 7]. Вычисляется многомерное расписание (multidimensional affine schedule), обладающее параллелизмом и сохраняющее логический порядок операций в соответствии с зависимостями по данным.

3. Поиск пространственного размещения [8, 9]. Вычисляется аффинное размещение (affine placement), обладающее параллелизмом, и уменьшающее стоимость коммуникаций.

4. Разбиение (tiling) индексного пространства виртуальных процессоров [9, 10] — сопоставление виртуальных процессоров физическим.

5. Генерация параллельной программы по алгоритму [11] в соответствии с пространственно-временным преобразованием (space-time mapping), заданным функциями расписания и размещения, и разбиением.

Широко применяемыми на практике стали классические методы поиска расписаний, ориентированные на минимизацию задержки расписания [6, 7], и современный подход [4], ориентированный на оптимизацию локальности данных. Подход, предложенный в работе [4] и реализованный в компиляторе Pluto, демонстрирует эффективность на практике [12]. Однако, метод, его реализующий, не лишен следующего конструктивного недостатка: поскольку задача сводится к поиску лексикографического минимума на многограннике [13], возникает вопрос о порядке следования переменных в целевом векторе, или, в терминах теории принятия решений, вопрос о предпочтениях ЛПР (лица, принимающего решение) относительно качества оптимизации для каждой зависимости 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.

Целью настоящего исследования является разработка метода поиска пространственно-временных преобразований (аффинных расписаний и размещений), основанного на той же идее оптимизации локальности данных, но предоставляющего более гибкие средства задания предпочтений ЛПР. Это успешно достигается с предложенной формулировкой задачи многокритериальной оптимизации, приводимой в дальнейшем к скалярной форме и решаемой методами линейного целочисленного программирования. Вычисляется многомерное расписание по жадной схеме, предложенной Футриером [7], и одномерное размещение, обладающее свойством вперед направленных коммуникаций (forward communication only placements) [9].

Вначале приводится обзор основных понятий модели многогранников, затем следует описание разработанного метода для вычисления оптимальных пространственно-временных преобразований. В завершение приводится пример распараллеливания LU-разложения и сравнение быстродействия с результатами компилятора Pluto.

1. Базовые понятия модели многогранников

Модель многогранников — модель последовательных и параллельных вычислений, рассматривающая класс линейных программ [14]. Эти программы удовлетворяют следующим ограничениям:

- единственным типом исполнительного оператора может быть оператор присваивания, правая часть которого является арифметическим выражением;
- все повторяющиеся операции описываются только с помощью циклов DO языка FORTRAN (либо эквивалентными циклами FOR языка C); структура вложенности циклов может быть произвольной; шаги изменения параметров циклов всегда равны +1;
- допускается использование 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 целочисленны; конкретные значения внешних переменных известны только во время работы программы.

Обобщенный граф зависимостей — это ориентированный мультиграф, каждая вершина которого представляет множество соответственных операций (динамических экземпляров одной и той же инструкции). 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.

Каждой вершине X соответствует ее домен — многогранник D_X на множестве Q^{p_X}, где p_X — размерность ее итерационного пространства (количество циклов в программе, включающих X). 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. Количество внешних переменных обозначим q_z.

Каждому ребру e, исходящему из вершины $\sigma(e) = X$ в $\delta(e) = Y$, ставится в соответствие многогранник R_e на множестве $\mathrm{Q}^{p_X + p_Y}$ 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 $\langle V; E; D; R \rangle$, где V — множество вершин, E — множество ребер, D — функция из множества вершин V во множество соответствующих доменов, R — функция из множества ребер E во множество соответствующих многогранников зависимостей.

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

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

Расписание для обобщенного графа зависимостей есть функция $\theta : \Omega \to \mathrm{N}_0$ такая, что

$$\forall e \in E(\vec{x} \in D_{\sigma(e)}, \vec{y} \in D_{\delta(e)}, \begin{bmatrix} \vec{x} \\ \vec{y} \end{bmatrix} \in R_e \Rightarrow \theta(\langle \vec{y}, \vec{z}; \delta(e) \rangle) \geq \theta(\langle \vec{x}, \vec{z}; \sigma(e) \rangle) + 1).$$

Размещение для обобщенного графа зависимостей есть функция $\pi : \Omega \to \mathrm{N}_0$, которая ставит в соответствие операции номер виртуального процессора, на котором она будет исполняться.

Назначением процедуры анализа потока данных [5] является упрощение представления R_e. Для каждой зависимости e имеем многогранник P_e и аффинное преобразование h_e такое, что:

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

Множество операций $F(t) = \{u \in \Omega \mid \theta(u) = t\}$ называется фронтом расписания на t. Программа с синхронным параллелизмом может быть сконструирована следующим образом:

```
do t = 0; L
  exchange data
  pardo F(t)
  barrier
end do
```

Здесь $L = \max_{u \in \Omega}(\theta(t))$ — задержка расписания.

2. Оптимальные пространственно-временные преобразования

2.1. Аффинные отображения. Критерий оптимальности

Одномерное аффинное отображение (affine mapping) для инструкции X есть функция вида

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

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

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

Рассмотрим функцию $\chi_e(\vec{y},\vec{z}) = \varphi_{\delta(e)}(\vec{y},\vec{z}) - \varphi_{\sigma(e)}(h_e(\vec{y}),\vec{z}), \quad \vec{y} \in P_e$. Если отображения $\varphi_{\delta(e)}(\vec{y},\vec{z})$ и $\varphi_{\sigma(e)}(h_e(\vec{y}),\vec{z})$ используются для логического времени (расписание вычислений), то $\chi_e(\vec{y},\vec{z})$ дает меру задержки использования данных, поскольку фронты расписания исполняются поочередно. Минимизация этой задержки приводит к улучшению временной локальности программы, так как уменьшается вероятность вытеснения данных из кэша к моменту их использования. Если же указанные отображения используются для привязки операций к процессорам (размещение вычислений), то $\chi_e(\vec{y},\vec{z})$ дает меру расстояния использования данных. Минимизация этого расстояния влечет размещение операций, индуцирующих информационную зависимость, на как можно более близких процессорах. Это позволяет уменьшить стоимость коммуникаций — например, вместо межузловой пересылки сообщений в кластерной системе в некоторых случаях можно обойтись межпроцессорной в рамках одного узла.

Предлагаемые далее методы поиска аффинных расписаний и размещений нацелены на минимизацию задержки и расстояния использования данных соответственно. Разницу $\chi_e(\vec{y},\vec{z})$ желательно ограничить постоянным значением. Однако, это не всегда возможно. Ответ на вопрос о верхней границе для $\chi_e(\vec{y},\vec{z})$ дает следующая теорема.

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

Доказательство. Поскольку h-преобразование является аффинным [5], то его можно задать матрицей $\mathbf{H}$:

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

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

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

Очевидно, что:

$$\chi_e(\vec{y},\vec{z}) = \vec{v}_{\delta(e)} \cdot \vec{y} - \vec{v}_{\sigma(e)} \cdot (\mathbf{H}_e^{\vec{y}} \vec{y} + \mathbf{H}_e^{\vec{z}} \vec{z} + \mathbf{H}_e^1) + (\vec{v}_{\delta(e)}^{'} - \vec{v}_{\sigma(e)}^{'}) \cdot \vec{z} + v_{\delta(e)}^0 - v_{\sigma(e)}^0 =$$
$$(\vec{v}_{\delta(e)}^T - \vec{v}_{\sigma(e)}^T \mathbf{H}_e^{\vec{y}}) \vec{y} + (\vec{v}_{\delta(e)}^{'T} - \vec{v}_{\sigma(e)}^{'T} - \vec{v}_{\sigma(e)}^T \mathbf{H}_e^{\vec{z}}) \vec{z} + v_{\delta(e)}^0 - v_{\sigma(e)}^0 - \vec{v}_{\sigma(e)}^T \mathbf{H}_e^1.$$

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

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

$$\begin{cases} \mathbf{A}_{\delta(e)} \begin{bmatrix} \vec{y} \\ \vec{z} \end{bmatrix} + \vec{b}_{\delta(e)} \geq 0 \\ \left[\mathbf{A}_{\sigma(e)}^{h_e(\vec{y})} \quad \mathbf{A}_{\sigma(e)}^{\vec{z}} \right] \begin{bmatrix} \mathbf{H}_e^{\vec{y}} \vec{y} + \mathbf{H}_e^{\vec{z}} \vec{z} + \mathbf{H}_e^1 \\ \vec{z} \end{bmatrix} + \vec{b}_{\sigma(e)} \geq 0 \end{cases} \Leftrightarrow$$

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

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:

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

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

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

Из теоремы 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 (1) как равенства.

Определение. Пусть $\{\phi\}_{E'}$ — множество всех возможных 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 $\phi \in \{\phi\}_{E'}$ называется оптимальным для E', если он минимизирует функционалы $f_e(\phi) = L_e(\vec{w}) = \vec{l}_e \cdot \vec{w} + l_e^0$, $e \in E'$ так, что

$$\neg \exists \phi' \in \{\phi\}_{E'} \left(\forall e \in E' \left(f_e(\phi') \leq f_e(\phi) \right) \wedge \exists e \in E' \left(f_e(\phi') < f_e(\phi) \right) \right).$$

Здесь $\vec{w}$ — заданный для переменных $\vec{l}_e$ вектор весовых коэффициентов.

Приходим к задаче многокритериальной оптимизации, где:

- Множество всех допустимых наборов $\{\phi\}_{E'}$ является множеством исходов и альтернатив одновременно.
- $f_e(\phi) \to min$, $e \in E'$ — критерии качества альтернативы ϕ.
- Оптимальное решение принадлежит множеству Парето.

Будем искать решение, применив технику линейной свертки, где α_e — весовые коэффициенты, отражающие предпочтения ЛПР относительно качества оптимизации для каждой зависимости по данным:

$$\sum_{e \in E'} \alpha_e f_e \to \min, \quad \alpha_e \geq 0. \tag{2}$$

Такая формулировка точнее отражает предпочтения ЛПР в силу большей гибкости относительно [4, 6, 7], и позволяет реализовать естественную эвристику: ЛПР заинтересовано в наилучшем качестве оптимизации для тех зависимостей, которые чаще возникают. Для программ со слабой параметризацией или в случае Just-In-Time компиляции, когда значения внешних переменных программы становятся известными и выбирается $\vec{w} = \vec{z}$, становится возможной приоритезация 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 [15], реализованного в библиотеке PolyLib [16]. В случае статической компиляции рекомендуется выбрать значения $\vec{w}[i]$, $i = 1 \ldots q_z$ как можно более близкие к вероятным $\vec{z}[i]$ исходя из априорной информации относительно контекста $\vec{z}$, а затем вычислить $\#P_e$ в соответствии с предполагаемыми значениями $\vec{z} = \vec{w}$. Далее будем уточнять формулировку задачи для отыскания расписаний и размещений в предположении, 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

Пусть $\theta(\langle \vec{x}, \vec{z}; S_i \rangle)$ — аффинная функция расписания для инструкции S_i, $\vec{x} \in D_{S_i}$. Рассмотрим функцию $\tau_{e_j}(\vec{y}, \vec{z}) = \theta(\langle \vec{y}, \vec{z}; \delta(e_j) \rangle) - \theta(\langle \vec{h}_{e_j}(\vec{y}), \vec{z}; \sigma(e_j) \rangle)$, $\vec{y} \in P_{e_j}$. Эта функция дает меру задержки использования данных, поскольку фронты расписания исполняются последовательно. Минимизация этой задержки приводит к улучшению временной локальности программы.

Будем искать расписание θ как оптимальный набор аффинных отображений $\theta(\langle \vec{x}, \vec{z}; S_i \rangle)$, $i = 1, \ldots, m$ для E. Определим множество исходов (оно совпадает со множеством альтернатив):

$$\Theta = \{\theta \mid (\theta(\langle \vec{x}, \vec{z}; S_i \rangle) = \vec{v}_{S_i} \cdot \vec{x} + \vec{v}'_{S_i} \cdot \vec{z} + v^0_{S_i}, \quad v^0_{S_i} \in \mathrm{Z}, \vec{v}_{S_i} \in \mathrm{Z}^{p_{S_i}}, \vec{v}'_{S_i} \in \mathrm{Z}^{q_z}, \quad i = 1, \ldots, m) \wedge$$
$$(\forall \vec{x} \in D_{S_i} : \theta(\langle \vec{x}, \vec{z}; S_i \rangle) \geq 0, \quad i = 1, \ldots, m) \wedge (\vec{y} \in P_{e_j} \Rightarrow \tau_{e_j}(\vec{y}, \vec{z}) \geq 1, \quad j = 1, \ldots, n)\}.$$

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:

$$\begin{cases} \tau_{e_j}(\vec{y}, \vec{z}) - 1 \geq 0, j = 1, \ldots, n \\ -\tau_{e_j}(\vec{y}, \vec{z}) + \vec{l}_{e_j} \cdot \vec{z} + l^0_{e_j} \geq 0, j = 1, \ldots, n \end{cases} \tag{3}$$

Первое ограничение обусловлено принципом причинности, второе задает лимит задержки использования, который требуется минимизировать. Оптимальное расписание минимизирует функцию стоимости

$$C(\vec{l}_{e_1}, l^0_{e_1}, \ldots, \vec{l}_{e_n}, l^0_{e_n}) = \sum_{j=1}^{n} \alpha_{e_j} f_{e_j} \to \min, j = 1, \ldots, n. \quad (4)$$

при ограничениях (3).

Линеаризуем систему ограничений (3) с помощью классической техники на основе леммы Фаркаша.

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

Для каждой инструкции S_i выписывается шаблон функции расписания, неотрицательной внутри ее домена, где $\mu_{S_i,*} \geq 0$ — множители Фаркаша, а $p_{D_{S_i}}$ неравенств $\vec{a}_{S_i,k} \cdot \begin{bmatrix} \vec{x}_{S_i} \\ \vec{z} \end{bmatrix} + b_{S_i,k} \geq 0$ определяют домен D_{S_i}:

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

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

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

В итоге система ограничений примет вид:

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

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 Парето-оптимального решения задачи многокритериальной оптимизации сводится к решению задачи линейного целочисленного программирования: минимизировать функцию стоимости (4) при линеаризованных ограничениях (3). Решение задачи ЛЦП может быть осуществлено, например, методом ветвей и границ, реализованным в библиотеке GNU Linear Programming Kit [17]. Подстановка значений μ в соответствующие шаблоны функций расписания для каждой инструкции дает решение задачи поиска расписания θ.

2.3. Многомерные аффинные расписания

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 [7], 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 сверткой критериев (2). 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:

$$\begin{cases} \tau_e(\vec{y},\vec{z}) - \varepsilon_e \geq 0, & e \in E' \\ 0 \leq \varepsilon_e \leq 1, & e \in E' \end{cases}. \tag{5}$$

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: $\sum_{e \in E'} \varepsilon_e \to \max$ при ограничениях (5). Задача решается методами линейного целочисленного программирования после линеаризации системы ограничений (классическая техника с применением леммы Фаркаша). Пусть $E'_1 = \{e \mid e \in E' \wedge \varepsilon_e = 1\}$ — множество ребер, соответствующих удовлетворенным зависимостям, $E'_0 = \{e \mid e \in E' \wedge \varepsilon_e = 0\}$ — 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. Свойство вперед направленных коммуникаций

Размещение обладает свойством вперед направленных коммуникаций тогда и только тогда, когда все направления коммуникаций содержатся в конусе $(0+, \ \ldots, \ 0+)$. Применение размещений, обладающих таким свойством, представляет практический интерес, поскольку позволяет в некоторых случаях вдвое уменьшить количество партнеров коммуникации. В работе Грибля [9] это свойство обсуждается подробно.

Пусть $\pi(\langle \vec{x}, \vec{z}; S_i \rangle)$ — аффинная функция размещения для инструкции S_i, $\vec{x} \in D_{S_i}$. Рассмотрим функцию $\rho_{e_j}(\vec{y}, \vec{z}) = \pi(\langle \vec{y}, \vec{z}; \delta(e_j) \rangle) - \pi(\langle \vec{h}_{e_j}(\vec{y}), \vec{z}; \sigma(e_j) \rangle)$, $\vec{y} \in P_{e_j}$. Эта функция дает меру расстояния использования данных. Минимизация этого расстояния снижает издержки межпроцессорной коммуникации.

Предложенный Грибле м метод, ставший классическим, предполагает четыре этапа:

1. Построение системы неравенств $\rho_{e_j}(\vec{y}, \vec{z}) \geq 0, j = 1, \ldots, n$, 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 [18], реализованного в библиотеке PolyLib [16].
4. Выбор экстремального луча, соответствующего предпочтительному размещению, в соответствии с заданной моделью стоимости (единственная вершина конуса соответствует нулевому решению, поэтому только экстремальные лучи представляют интерес).

Можно выбрать оптимальное решение по этой схеме, применив предложенную модель стоимости (2). Однако это влечет полный перебор экстремальных лучей и трудоемкую оценку качества соответствующих им размещений непосредственным применением теоремы 1. Предпочтительнее искать аффинное размещение π как оптимальный набор аффинных отображений $\pi(\langle \vec{x}, \vec{z}; S_i \rangle)$, $i = 1, \ldots, m$ для E. Определим множество исходов (оно совпадает со множеством альтернатив):

$$\Pi = \{\pi \mid (\pi(\langle \vec{x}, \vec{z}; S_i \rangle) = \vec{v}_{S_i} \cdot \vec{x} + \vec{v}'_{S_i} \cdot \vec{z} + v^0_{S_i}, \quad v^0_{S_i} \in \mathrm{Z}, \vec{v}_{S_i} \in \mathrm{Z}^{p_{S_i}}, \vec{v}'_{S_i} \in \mathrm{Z}^{q_z}, \quad \vec{v}_{S_i} \neq \vec{0}, \quad i = 1, \ldots, m) \wedge$$
$$(\forall \vec{x} \in D_{S_i} : \pi(\langle \vec{x}, \vec{z}; S_i \rangle) \geq 0, \quad i = 1, \ldots, m) \wedge$$
$$(\pi(\langle \vec{x}, \vec{z}; S_i \rangle) \neq \vec{\gamma}_{S_i} \cdot \vec{\theta}(\langle \vec{x}, \vec{z}; S_i \rangle) + \vec{\gamma}'_{S_i} \cdot \vec{z} + \gamma^0_{S_i}, \quad \gamma^0_{S_i} \in \mathrm{Q}, \vec{\gamma}_{S_i} \in \mathrm{Q}^d, \vec{\gamma}'_{S_i} \in \mathrm{Q}^{q_z}, \quad i = 1, \ldots, m) \wedge$$
$$(\vec{y} \in P_{e_j} \Rightarrow \rho_{e_j}(\vec{y}, \vec{z}) \geq 0, \quad j = 1, \ldots, n)\}.$$

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 $\pi(\langle \vec{x}, \vec{z}; S_i \rangle) = \vec{\gamma}_{S_i} \cdot \vec{\theta}(\langle \vec{x}, \vec{z}; S_i \rangle) + \vec{\gamma}'_{S_i} \cdot \vec{z} + \gamma^0_{S_i}, \quad \gamma^0_{S_i} \in \mathrm{Q}, \vec{\gamma}_{S_i} \in \mathrm{Q}^d, \vec{\gamma}'_{S_i} \in \mathrm{Q}^{q_z}$. 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 [19], позволяющий задать линейные ограничения, фиксирующие линейную независимость и исключающие нулевое решение из рассмотрения при выборе альтернатив. Дальнейшие рассуждения опираются на следующий факт:

$$\begin{cases} c_i \in \mathrm{Z}, \quad i = 1,\ldots,p \\ b \geq 2, \quad b \in \mathrm{Z} \\ -b < c_i < b, \quad i = 1,\ldots,p \\ \left[c_1 \quad \ldots \quad c_p\right]^T \neq \vec{0} \end{cases} \Leftrightarrow \begin{cases} c_i \in \mathrm{Z}, \quad i = 1,\ldots,p \\ b \geq 2, \quad b \in \mathrm{Z} \\ -b < c_i < b, \quad i = 1,\ldots,p \\ \varepsilon \in \{0,1\} \\ \sum\limits_{i=1}^{p} b^{i-1} c_i \geq 1 - \varepsilon b^{p+1} \\ -\sum\limits_{i=1}^{p} b^{i-1} c_i \geq 1 - (1-\varepsilon) b^{p+1} \end{cases}. \tag{6}$$

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 $b \geq 2$ и немедленно получить линейные ограничения, исключающие нулевое решение из рассмотрения. В работе [19] предлагается значение $b = 5$, как достаточное для большинства практических случаев.

Пусть $\mathbf{H}_{S_i}$ — матрица, строки которой представляют компоненты ранее найденного многомерного аффинного расписания для инструкции S_i: $\mathbf{H}_{S_i} = \left[\vec{v}_{S_i,1} \quad \cdots \quad \vec{v}_{S_i,d}\right]^T$. Обозначим $\mathbf{H}_{S_i}^{\perp}$ подпространство ортогональное $\mathbf{H}_{S_i}$, т.е., любая строка $\mathbf{H}_{S_i}^{\perp}$ ортогональна любой строке $\mathbf{H}_{S_i}$ ($\mathbf{H}_{S_i}^{\perp}\mathbf{H}_{S_i}^{T} = \mathbf{0}$). Вычислить ортогональное подпространство можно следующим образом: $\mathbf{H}_{S_i}^{\perp} = \mathbf{I} - \mathbf{H}_{S_i}^{T}(\mathbf{H}_{S_i}\mathbf{H}_{S_i}^{T})^{-1}\mathbf{H}_{S_i}$ [4].

Пусть $\mathbf{H}_{S_i}^{\perp}[r]$ — r-я строка матрицы $\mathbf{H}_{S_i}^{\perp}$, $r = 1,\ldots,p_{S_i}$. 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 $\mathbf{H}_{S_i}^{\perp}$:

$$\left[\mathbf{H}_{S_i}^{\perp}[1]\vec{v}_{S_i} \quad \ldots \quad \mathbf{H}_{S_i}^{\perp}[p_{S_i}]\vec{v}_{S_i}\right]^T \neq \vec{0}. \tag{7}$$

Если установлены ограничения на компоненты $\vec{v}_{S_i}$ с некоторым параметром $b \geq 2$, то компоненты вектора в условии (7) являются ограниченными:

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

Аналогичным образом применяется утверждение (6) к условию (7) для получения ограничений линейной независимости. Параметр B выбирается следующим образом:

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

Поиск экстремумов здесь может быть выполнен методами линейного целочисленного программирования. Скомбинируем все ограничения на функции размещения в одну систему:

$$\begin{cases} \rho_{e_j}(\vec{y}, \vec{z}) \geq 0, \quad j = 1,\ldots,n \\ -\rho_{e_j}(\vec{y}, \vec{z}) + \vec{l}_{e_j} \cdot \vec{z} + l_{e_j}^0 \geq 0, \quad j = 1,\ldots,n \\ \vec{v}_{S_i} \neq \vec{0}, \quad i = 1,\ldots,m \\ \left[\mathbf{H}_{S_i}^{\perp}[1]\vec{v}_{S_i} \quad \ldots \quad \mathbf{H}_{S_i}^{\perp}[p_{S_i}]\vec{v}_{S_i}\right]^T \neq \vec{0}, \quad i = 1,\ldots,m \end{cases}. \tag{8}$$

Первое ограничение обеспечивает свойство вперед направленных коммуникаций, второе задает лимит расстояния использования, который требуется минимизировать, третье исключает из рассмотрения нулевое решение, четвертое обеспечивает линейную независимость от компонентов многомерного аффинного расписания. Оптимальное размещение минимизирует функцию стоимости (4) при ограничениях (8). Задача решается методами линейного целочисленного программирования после линеаризации первого и второго ограничений в системе (8) 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 или ускорителя Intel Xeon PHI. Одномерная вычислительная сетка в терминах спецификаций OpenCL и CUDA успешно моделируется таким же образом при взаимно однозначном соответствии между вычислительной нитью и виртуальным процессором.

3. Пример: распараллеливание LU-разложения

Рассмотрим алгоритм LU-разложения квадратной матрицы A:

```
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 четыре его параллельные реализации, полученные различными способами: распараллеливание вручную, компиляция Pluto, и две реализации, полученные с помощью предложенного метода. Для обозначения параллелизма циклов используются директивы стандарта OpenMP. Поскольку циклы по l и по i являются параллельными и их слияние допустимо, то очевидное распараллеливание можно произвести вручную:

```
#pragma omp parallel
{
  for (int k = 0; k < N; k++) {
    #pragma omp for
    for (int i = k + 1; i < N; i++) {
      A[i][k] /= A[k][k];
      for (int j = k + 1; j < N; j++)
        A[i][j] -= A[i][k] * A[k][j];
    }
  }
}
```

Во всех последующих случаях для генерации параллельной программы используется библиотека CLooG [11] версии 0.18.2. Используются следующие директивы препроцессора языка C:

```
#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)))
```

Компилятор Pluto версии 0.11.3 дает результат с многомерными аффинными отображениями $\Phi_{S_1} = \begin{bmatrix} k+l \\ l \\ k \end{bmatrix}$ и $\Phi_{S_2} = \begin{bmatrix} k+i \\ i \\ j \end{bmatrix}$, где второе измерение используется для пространства процессоров:

```
#pragma omp parallel
{
  for (int p1=1;p1<=2*N-3;p1++) {
    #pragma omp for
    for (int p2=ceild(p1+1,2);p2<=min(p1,N-1);p2++) {
      S1((p1-p2),p2);
      for (int p3=p1-p2+1;p3<=N-1;p3++)
        S2((p1-p2),p2,p3);
    }
  }
}
```

Априори известно, что параметр N задает размер задачи и должен быть большим положительным числом, поэтому целесообразно выбрать $\vec{w} = [1000]$. Оптимизация предложенным методом с соблюдением всех рекомендаций дает $\theta_{S_1} = 2k$, $\theta_{S_2} = 2k+1$, $\pi_{S_1} = l-1$, $\pi_{S_2} = i-1$:

```
#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);
    }
  }
}
```

В пессимистическом случае, если ничего неизвестно о контексте, выбирается $\vec{w} = [1]$ и $\alpha_{e_j} = 1$ для всех e_j. Это дает $\theta_{S_1} = 2k$, $\theta_{S_2} = k+j$, $\pi_{S_1} = l-1$, $\pi_{S_2} = i-1$:

```
#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));
    }
  }
}
```

Эксперименты проводились для матрицы размера 2048×2048 вещественных чисел двойной точности на восьмиядерном процессоре Intel Xeon E2690 под операционной системой Linux CentOS 6.5 x64. Компиляция производилась с помощью gcc версии 4.4.7 с флагом оптимизации O3. В Табл. 1 представлены значения ускорения, достигаемого при параллельном выполнения LU-разложения в 8 нитей относительно изначального последовательного варианта.

Табл. 1. Ускорение от распараллеливания LU-разложения

Вручную	Pluto	Оптимально	Пессимистично
7,38	6,47	7,31	0,94

Лучшее быстродействие достигается при распараллеливании вручную, так как итоговый код содержит меньше всего управляющих конструкций. Вариант, полученный с помощью предложенного метода и с соблюдением всех рекомендаций, дает очень близкое к наилучшему быстродействие и оказывается на 13% быстрее варианта Pluto. Неудовлетворительный результат дает оптимизация в пессимистических условиях: производительность страдает от кэш-промахов.

Даже в случаях неопределенности 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 естественному соображению о предпочтительности ограничения задержек и расстояний использования данных постоянным числом. Также не рекомендуется рассмотрение всех зависимостей как равновесных. Всегда можно выбрать положительные значения больше 1 для компонентов $\vec{w}$, а также размерности многогранников зависимостей P_{e_j} для весов α_{e_j}. Применение подобных рекомендаций в рассматриваемом случае приводит пессимистический вариант параллельного LU-разложения к виду оптимального (выбрано $\alpha_{e_j} = \# P_{e_j}$ и $\vec{w} = [1000]$).

Заключение

В ходе исследования была показана необходимость совершенствования способа задания предпочтений ЛПР для того, чтобы повысить качество вычисляемых пространственно-временных преобразований при автоматическом распараллеливании линейных программ. Предложен метод поиска оптимальных преобразований, нацеленный на улучшение временной локальности данных и снижение издержек межпроцессорной коммуникации. Рассмотрение задач поиска расписаний и размещений как задач многокритериального выбора в условиях полной определенности позволяет задать линейную свертку критериев качества, точнее отражающую предпочтения ЛПР, и потому позволяет находить более удачные (в плане повышения быстродействия синтезированной параллельной программы) пространственно-временные преобразования, чем классические методы, опирающиеся на технику поиска лексикографического минимума на многограннике. Наилучшие результаты достигаются для программ со слабой параметризацией или программ с искусственно ослабленной параметризацией во время Just-In-Time компиляции. 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. Проблема оптимальности многомерного аффинного преобразования остается открытой и является предметом дальнейших исследований.

Литература

1. Lengauer C. Loop parallelization in the polytope model //CONCUR'93. – Springer Berlin Heidelberg, 1993. – С. 398-416.
2. Irigoin F., Jouvelot P., Triolet R. Semantical interprocedural parallelization: An overview of the PIPS project //Proceedings of the 5th international conference on Supercomputing. – ACM, 1991. – С. 244-251.
3. Griebl M., Lengauer C. The loop parallelizer LooPo //Proc. Sixth Workshop on Compilers for Parallel Computers. – Konferenzen des Forschungszentrums Jülich, 1996. – Т. 21. – С. 311-320.
4. Bondhugula U. et al. Automatic transformations for communication-minimized parallelization and locality optimization in the polyhedral model //Compiler Construction. – Springer Berlin Heidelberg, 2008. – С. 132-146.
5. Feautrier P. Dataflow analysis of array and scalar references //International Journal of Parallel Programming. – 1991. – Т. 20. – №. 1. – С. 23-53.
6. Feautrier P. Some efficient solutions to the affine scheduling problem. I. One-dimensional time //International journal of parallel programming. – 1992. – Т. 21. – №. 5. – С. 313-347.

7. Feautrier P. Some efficient solutions to the affine scheduling problem. Part II. Multidimensional time //International journal of parallel programming. – 1992. – Т. 21. – №. 6. – С. 389-420.
8. Feautrier P. Toward automatic distribution //Parallel Processing Letters. – 1994. – Т. 4. – №. 03. – С. 233-244.
9. Griebl M. Automatic parallelization of loop programs for distributed memory architectures. – Univ. Passau, 2004.
10. Bandishti V., Pananilath I., Bondhugula U. Tiling stencil computations to maximize parallelism //Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. – IEEE Computer Society Press, 2012. – С. 40.
11. Bastoul C. Code generation in the polyhedral model is easier than you think //Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques. – IEEE Computer Society, 2004. – С. 7-16.
12. Bondhugula, Uday, et al. "A practical automatic polyhedral parallelizer and locality optimizer." ACM SIGPLAN Notices 43.6 (2008): 101-113.
13. Feautrier P. Parametric integer programming //RAIRO Recherche opérationnelle. – 1988. – Т. 22. – №. 3. – С. 243-268.
14. Воеводин В. В. Параллельные вычисления. – БХВ-Петербург, 2004.
15. Clauss P. Counting solutions to linear and nonlinear constraints through Ehrhart polynomials: Applications to analyze and transform scientific programs //25th Anniversary International Conference on Supercomputing Anniversary Volume. – ACM, 2014. – С. 237-244.
16. Loechner V. PolyLib: A library for manipulating parameterized polyhedra. – 1999.
17. Makhorin A. GNU linear programming kit. Moscow Aviation Institute, Moscow, Russia. – 2012.
18. Черникова Н. В. Алгоритм для нахождения общей формулы неотрицательных решений системы линейных неравенств //Журнал вычислительной математики и математической физики. – 1965. – Т. 5. – №. 2. – С. 334-337.
19. Bondhugula U., Cohen A. Handling Negative Coefficients in Automatic Transformation Schedules. – 2014.

Лебедев Артем Сергеевич. Аспирант Рыбинского государственного авиационного технического университета. Окончил Рыбинский государственный авиационный технический университет в 2012 году. Автор 8 печатных работ. Область научных интересов: параллельное программирование, оптимизирующие преобразования программ, вычислительная математика. E-mail**:** tementy@gmail.com.