Влияние полиэдральной оптимизации на локальность использования данных при распараллеливании линейных программ на примере atax

А. С. Лебедев

***Аннотация*—В статье рассматривается влияние полиэдральной оптимизации на важное свойство программы — локальность использования данных, которое тесно связано с возможным ограничением ее производительности как при последовательном, так и параллельном выполнении. Предложен новый способ нахождения многомерных расписаний линейных программ, комбинирующий топологическую сортировку вершин обобщенного графа зависимостей и жадную схему Футриера с ранее предложенными модификациями. Приводятся экспериментальные результаты исследования производительности параллельных вариантов программы atax из набора тестов polybench, полученных применением современного транслятора pluto и разработанного автором транслятора ilpy, что позволяет сравнить два подхода к постановке задач оптимизации при выборе аффинных отображений: лексикографическая оптимизация (pluto) и линейное целочисленное программирование (ilpy). С помощью средств профилирования oprofile/ocount выявлена неэффективность использования кэшей многоядерного процессора и показана необходимость совершенствования критериев оптимальности пространственных и временных отображений линейных программ.**

***Ключевые слова*—Линейная программа, локальность использования данных, модель многогранников, профилирование.**

I. Введение

Оптимальное использование вычислительных ресурсов всегда представляло сложную научную задачу, а с учетом современных тенденций внедрения высокопроизводительных гетерогенных вычислительных систем ее актуальность поддерживается новыми требованиями к эффективности программирования параллельных архитектур, в особенности с накопленной многолетней кодовой базой.

В литературных источниках отмечается, что большинство инженерных и научных приложений тратят основную часть времени выполнения на участки с циклическими конструкциями [1, 2]. Зачастую эти конструкции удовлетворяют критериям линейности программ [3, с. 340], которые представляют для исследователей особый интерес, так как могут быть проанализированы с помощью известных математических методов, опирающихся на глубокие теоретические результаты в области выпуклого анализа и линейного целочисленного программирования.

Модель многогранников [4] широко используется в статической компиляции [5—8] для анализа и трансформации программ линейного класса. Практический эффект ее применения (полиэдральной оптимизации программ) заключается в повышении степени параллелизма и улучшения локальности использования данных [3, с. 57; 9, с. 911, 920, 1039], как пространственной, так и временной. Это достигается за счет выбора расписаний и размещений вычислений (а также размещений данных для систем с распределенной памятью) в согласии с заданными критериями оптимальности, выражающими некоторое эвристическое соображение, и постановка задачи оптимизации разнится в зависимости от применяемой целевой функции. Целью работы является выявление неэффективности использования кэша многоядерного процессора при выполнении параллельных программ, полученных в результате полиэдральной оптимизации.

В разделе II приводятся базовые понятия модели многогранников, рассматривается наиболее распространенная для оптимизирующих трансляторов лексикографическая оптимизация, применяемая современным инструментом pluto [8], и ранее предложенный автором подход [10], основанный на оптимизации методами линейного целочисленного программирования, оперирующий взвешенной суммой показателей качества решения, и доведенный до практического воплощения в трансляторе ilpy [11]. Также предложен новый способ нахождения многомерного расписания вычислений, основанный на комбинации топологической сортировки вершин обобщенного графа зависимостей [12] и жадной схемы Футриера [13] с модификациями [10]. Предложенный способ вдохновлен структурой обобщенного графа зависимостей программы atax из набора тестов polybench [14], которая рассматривается как тестовый пример в настоящей работе.

В разделе III приводятся экспериментальные

Статья получена 3 апреля 2024.

Лебедев Артем Сергеевич, МИРЭА – Российский технологический университет, (e-mail: tementy@gmail.com).

исследования производительности параллельных вариантов программы atax, полученные применением трансляторов pluto и ilpy.

В разделе IV приводятся методика и результаты профилирования параллельных программ с применением средства oprofile/ocount [15] на вычислительной машине с процессором Intel с архитектурой Ivy Bridge.

В заключении обсуждаются причины ухудшения производительности параллельных программ и приводятся направления дальнейших исследований.

II. Материалы и методы

A. *Базовые понятия модели многогранников*

Обобщенный граф зависимостей — это ориентированный мультиграф, каждая вершина которого представляет множество соответственных операций (динамических экземпляров одной и той же инструкции) [12]. 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.

Каждой вершине X соответствует ее домен — многогранник D_X на множестве $\mathbb{Q}^{p_X}$, 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, имеющий размерность q_z. 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 $\begin{bmatrix} \vec{i} \\ \vec{j} \end{bmatrix} \in R_e$.

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

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

Пусть E — множество ребер обобщенного графа зависимостей. Тогда d-мерное $(d \geq 1)$ расписание вычислений есть функция $\theta : \Omega \to \mathbb{Z}_0^d$ такая, что

$$\forall e \in E, \vec{x} \in D_{\sigma(e)}, \vec{y} \in D_{\delta(e)} : \begin{bmatrix} \vec{x} \\ \vec{y} \end{bmatrix} \in R_e \Rightarrow$$
$$\theta(\vec{y}, \vec{z}; \delta(e)) >_{lex} \theta(\vec{x}, \vec{z}; \sigma(e)).$$

Множество операций $F(t) = \{u \in \Omega \mid \theta(u) = t\}$ называется фронтом расписания на t (или ярусом параллельной формы [3, с. 194]).

Размещение вычислений есть функция $\pi : \Omega \to \mathbb{Z}_0$, которая ставит в соответствие операции номер виртуального процессора, на котором она будет исполняться.

Одномерное аффинное отображение для инструкции X есть функция вида $\varphi_X(\vec{i}, \vec{z}) = \vec{v}_X \cdot \vec{i} + \vec{v}'_X \cdot \vec{z} + v^0_X$, $v^0_X \in \mathbb{Z}$, $\vec{v}_X \in \mathbb{Z}^{p_X}$, $\vec{v}'_X \in \mathbb{Z}^{q_z}$, $\vec{i} \in D_X$. Многомерное аффинное отображение для инструкции X задается конечным набором (вектором) одномерных отображений.

B. *Лексикографическая оптимизация в pluto*

Бондхугулой были развиты идеи Футриера оптимизации временной локальности использования данных [12], и реализованы без минимизации размерности расписания в распараллеливающем трансляторе pluto [8]. Для каждой инструкции вычисляется столько же легальных отображений φ, сколько их существует в исходной программе:

$$\begin{bmatrix} \vec{x} \\ \vec{y} \end{bmatrix} \in R_e \Rightarrow 0 \leq \varphi_{\delta(e)}(\vec{y}, \vec{z}) - \varphi_{\sigma(e)}(\vec{x}, \vec{z}) \leq L_e(\vec{z}),$$

где $L_e(\vec{z})$ — аффинная функция, зависящая только от внешних переменных программы. Предполагается минимизация этой верхней границы: коэффициенты $L_e(\vec{z})$ для каждого рассматриваемого ребра e вписываются в целевой вектор и осуществляется лексикографическая оптимизация для нахождения решения [16]. При этом все вычисляемые отображения линейно не зависимы. 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 автором метод [10] нахождения пространственных и временных отображений минимизируют задержку и расстояние использования данных, снимая указанное ограничение.

C. *Целевые функции в ilpy*

Автором были развиты идеи Футриера [12], Бондхугулы [5] и Грибля [17] для нахождения аффинных отображений линейных программ [10], и реализованы в трансляторе ilpy [11]. Компоненты многомерного расписания и одномерное размещение вычислений

формируются отдельно, и удовлетворяют условиям:

$$\begin{bmatrix}\vec{x}\\ \vec{y}\end{bmatrix} \in R_e \Rightarrow 1 \le \theta^{(\cdot)}\left(\langle \vec{y}, \vec{z}; \delta(e)\rangle\right) - \theta^{(\cdot)}\left(\langle \vec{x}, \vec{z}; \sigma(e)\rangle\right) \le L_e^{(\cdot)}(\vec{z}),$$

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

Оптимальное аффинное отображение для подмножества ребер $E' \subseteq E$ минимизирует целевую функцию $C = \sum_{e \in E'} \alpha_e L_e^*(\vec{z})$, где $\alpha_e \ge 0$ — весовые коэффициенты, рассматриваемые как показатели относительной значимости зависимостей по данным. В трансляторе применяется $\alpha_e = \#R_e$, где $\#R_e$ — количество точек с целочисленными координатами внутри многогранника зависимостей R_e, а $\vec{z}$ оценивается исходя из априорной информации относительно контекста. Для внешних переменных программы, задающих размер задачи, по умолчанию используется оценка 100, если значение не задано явно.

D. Алгоритм atax и его реализации

Матричное произведение atax подразумевает вычисление $y = A^T(Ax)$, где A — матрица размера $M \times N$, x и y — векторы длины N.

Рассматривается программа из набора тестов polybench [14] (рис. 1). В дальнейшем данный вариант будет называться atax_p. Также рассматривается модифицированный, но сохраняющий корректность вычислений вариант программы (рис. 2). Он требует меньше памяти для выполнения, но убирает возможность параллельного вычисления элементов массива `tmp`: массив заменяется на переменную. Домены инструкций не претерпели изменений. В дальнейшем данный вариант будет называться atax.

```
#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
```

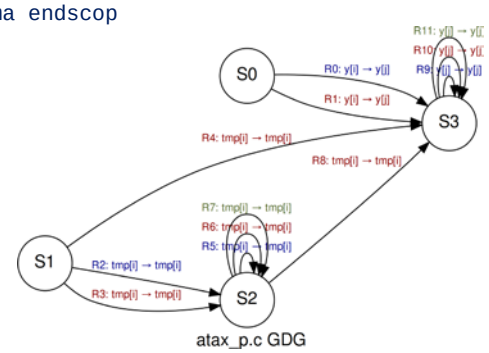

Рис. 1. Код на языке C и обобщенный граф зависимостей atax_p.

Для программы atax_p (рис. 1) 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».

```
#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
```

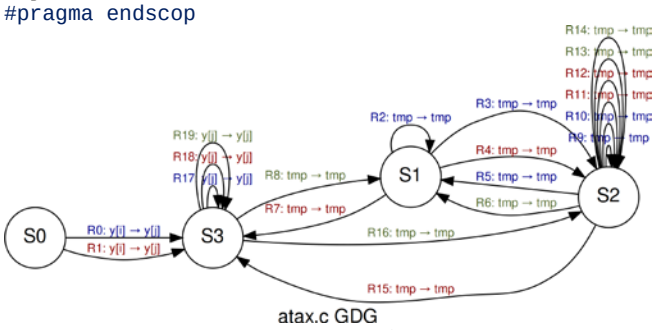

Рис. 2. Код на языке C и обобщенный граф зависимостей atax.

Для программы atax (рис. 2) 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».

E. Топологическая сортировка вершин обобщенного графа зависимостей

Рассмотрим частный случай, когда обобщенный граф зависимостей программы не содержит циклов, при этом могут присутствовать петли (пример на рис. 1). 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 [18, 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 $ord : \{S_i \mid i = 1, \ldots, m\} \to \mathbb{Z}_0$ — 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 [10].

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

1. Если задан параметр --smdp, то
   - 1.1. $E'_{net} \leftarrow \{e \mid e \in E \wedge \sigma(e) \neq \delta(e)\}$;
     $E'_{loop} \leftarrow \{e \mid e \in E \wedge \sigma(e) = \delta(e)\}$.
   - 1.2. Выполнить алгоритм Демукрона для графа $\langle \{S_i \mid i = 1,\ldots,m\}; E'_{net} \rangle$ 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 [18, с. 354].
   - 1.3. $\vec{\theta}^{(1)}_{S_i} \leftarrow ord(S_i), \quad i = 1,\ldots,m$.
   - 1.4. 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.

III. Экспериментальные исследования производительности параллельных вариантов 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 линейного целочисленного программирования 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:

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

Достаточно одномерного расписания для удовлетворения всех информационных зависимостей в atax_p, и размещение вычислений обладает свойством вперед направленных коммуникаций [17].

Найденное расписание позволяет ограничить задержку использования данных для шести из двенадцати ребер в обобщенном графе зависимостей постоянными величинами:

```
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
```

Найденное размещение вычислений позволяет ограничить расстояние использования данных постоянной величиной для восьми из двенадцати ребер в обобщенном графе зависимостей:

```
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
```

Параллельный вариант atax_p, сгенерированный ilpy, представлен на листинге 1.

Транслятор pluto выдает программный код (листинг 2) в соответствии со следующими пространственно-временными отображениями:

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

Вычисленные ilpy для atax аффинные отображения имеют вид:

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

$$\theta_{S_1} = \begin{bmatrix} 3i+1 \\ 0 \end{bmatrix}; \quad \pi_{S_1} = [i];$$

$$\theta_{S_2} = \begin{bmatrix} 3i+2 \\ j \end{bmatrix}; \quad \pi_{S_2} = [i];$$

$$\theta_{S_3} = \begin{bmatrix} 3i+3 \\ 0 \end{bmatrix}; \quad \pi_{S_3} = [i - j + N].$$

Потребовалось двумерное расписание для atax, и размещение вычислений не обладает свойством вперед направленных коммуникаций.

Листинг 1. Параллельный вариант atax_p_ilp_sync, синхронный параллелизм.

```
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];
        }
        if (ilpp <= floord(N-2,2)) {
          y[-ilpp+N-1] = 0;
        }
      }
      #pragma omp master
      for (int ilpp=N;ilpp<=N;ilpp++) {
        tmp[N] = 0;
      }
    }
    if ((M >= 1) && (N >= M)) {
      #pragma omp barrier
      #pragma omp for
      for (int 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;
        }
      }
      #pragma omp barrier
      #pragma omp for
      for (int ilpp=M;ilpp<=N-1;ilpp++) {
        y[-ilpp+N-1] = 0;
      }
    }
    if (N >= 1) {
      for (int t0=N+1;t0<=M-1;t0++) {
        #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];
        }
        #pragma omp master
        for (int ilpp=t0;ilpp<=t0;ilpp++) {
          tmp[t0] = 0;
        }
      }
    }
    for (int t0=max(M,N+1);t0<=N+M-1;t0++) {
      #pragma omp barrier
      #pragma omp for
      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 2. Параллельный вариант atax_p_pluto, синхронный параллелизм.

```
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 */
  }
}
```

Найденное расписание обладает следующими свойствами: первый компонент удовлетворяет семнадцать зависимостей из двадцати и для пятнадцати из них позволяет ограничить задержку использования данных постоянной величиной; второй компонент удовлетворяет оставшиеся три зависимости, и для каждой из них позволяет ограничить задержку использования данных постоянной величиной.

```
  Trying to find a schedule component 1
  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
  3 deps satisfied, 0 deps unsatisfied with 2
dimension
  Trying to find a schedule component 2
  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
  Schedule has 2 dimensions
```

Найденное размещение вычислений позволяет ограничить расстояние использования данных постоянной величиной для четырнадцати из двадцати ребер в обобщенном графе зависимостей.

```
  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
```

Параллельный вариант atax, сгенерированный ilpy, представлен на листинге 3.

Транслятор pluto выдает программный код (листинг 4) в соответствии со следующими пространственно-временными отображениями:

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

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:

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

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

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.

Листинг 3. Параллельный вариант atax_ilp_sync, синхронный параллелизм.

```
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 4. Параллельный вариант atax_pluto, синхронный параллелизм.

```
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 */
```

```
  }
}
```

Второй компонент расписания позволяет ограничить задержку использования данных для всех оставшихся шести ребер в обобщенном графе зависимостей постоянными величинами.

```
  6 deps satisfied, 6 deps unsatisfied with 1 dimension
  Trying to find a schedule component 1
  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
  6 deps satisfied, 0 deps unsatisfied with 2 dimension
  Trying to find a schedule component 2
  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
  Schedule has 2 dimensions
```

Найденное размещение вычислений позволяет ограничить расстояние использования данных постоянной величиной для восьми из двенадцати ребер в обобщенном графе зависимостей.

```
  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
```

Параллельный вариант atax_p, сгенерированный ilpy с применением топологической сортировки вершин обобщенного графа зависимостей, представлен на листинге 5.

Листинг 5. Параллельный вариант atax_p_ilp_sync_mdp, синхронный параллелизм.

```
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];
          }
        }
      }
    }
  }
```

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. Все тестовые запуски выполнялись на оборудовании РТУ МИРЭА: вычислительная машина на базе процессора Intel(R) Xeon(R) CPU E5-2690 v2 @ 3.00GHz, 16Гб оперативной памяти стандарта DDR3. Операционная система — Linux CentOS 7.8 x64. Компилятор — gcc версии 4.8.5, уровень оптимизации — наивысший -O3.

Значения параметров: `N` задано 6144, `M` задано 8192. Линеаризация матриц — по строкам, вычисления производились с двойной точностью. Графики ускорения относительно запуска последовательного варианта atax_p, а также сравнение параллельных вариантов приведены на рис. 3 и рис. 4 соответственно.

Параллельный вариант atax_p_ilp_sync_mdp показывает лучшую производительность во всех запусках, выигрывая от высокой доли параллельно исполняемых итераций циклов, включающих минимум используемых массивов и имеющих свойство локальности: два соседних виртуальных процессора обрабатывают по возможности два соседних элемента массива, для матриц подразумеваются действия в рамках одной строки. Это позволяет эффективно применять кэш процессора, содержащий фрагмент линейного массива или строку матрицы. Вариант atax_ilp_sync отрабатывает с меньшей производительностью, поскольку из-за модификации теряет долю независимых операций, но сохраняет

свойство локальности. Вариант atax_p_ilp_sync выполняет параллельное вычисление элементов массива `tmp`, но при этом каждый виртуальный процессор читает различные строки матрицы `A`, что увеличивает вероятность вытеснения нужных данных из кэша при обработке всех итераций параллельного цикла.

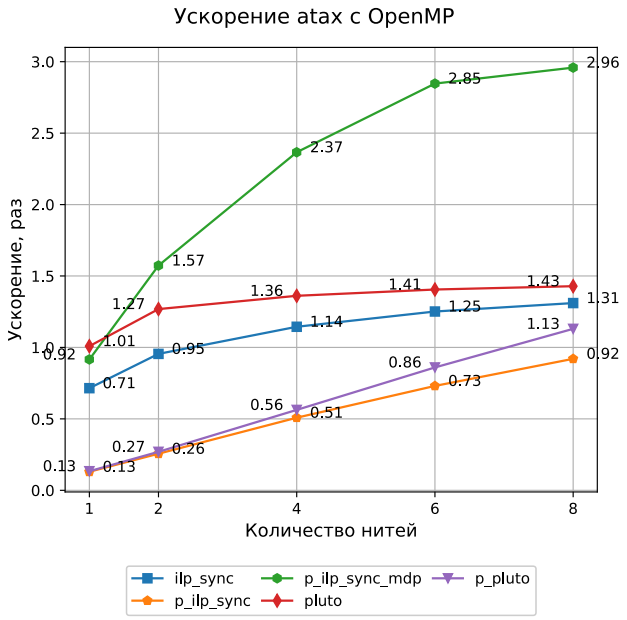

Рис. 3. Ускорение atax с OpenMP.

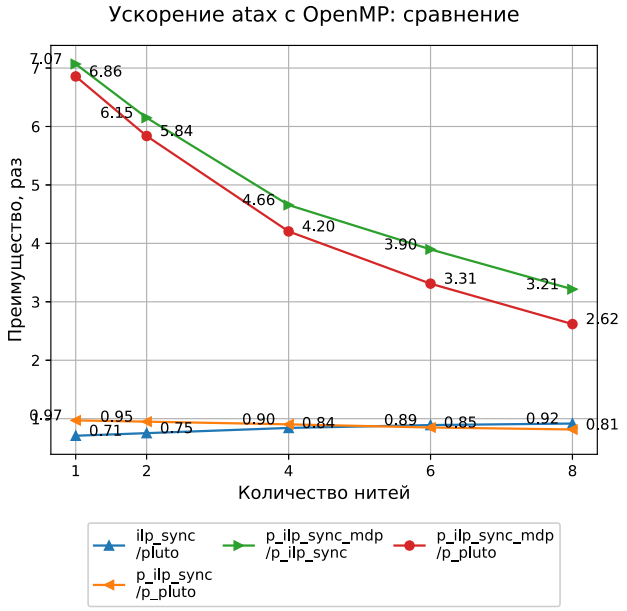

Рис. 4. Сравнение производительности параллельных вариантов atax с OpenMP.

Вариант atax_p_pluto имеет нарушения локальности в вычислении `y` при доступе к различным строкам матрицы `A` в рамках одной итерации параллельного цикла, но в силу отдельного эффективного вычисления `tmp` выигрывает у atax_p_ilp_sync. Вариант atax_pluto имеет сходную с atax_ilp_sync последовательность вычисления массивов, но последний сканирует массив `y` в обратном порядке и имеет большие вычислительные расходы в реализации расписания вычислений.

IV. Профилирование параллельных вариантов atax

A. *Методика профилирования приложений*

Для профилирования применялось средство oprofile [15] версии 1.4.0. Поскольку профилировщик является семплирующим, не требуется специальная перекомпиляция приложения. С помощью утилиты ocount были собраны значения аппаратных счетчиков событий, релевантных применяемой микроархитектуре Intel Ivy Bridge [19]:

1. CPU_CLK_UNHALTED — число тактов процессора.
2. l2_rqsts:all_demand_data_rd — количество запросов данных, адресованных в кэш второго уровня.
3. l2_rqsts:demand_data_rd_hit — количество запросов данных, адресованных в кэш второго уровня и обслуженных с кэш-попаданием.
4. LLC_MISSES — количество запросов данных, адресованных в кэш последнего уровня и увенчавшихся кэш-промахом.
5. LLC_REFS — количество запросов данных, адресованных в кэш последнего уровня.
6. mem_load_uops_llc_hit_retired:xsnp_hit — обработанные [20, с. 46] микрооперации загрузки, источники данных для которых были отображены в общий кэш последнего уровня, обслуженные благодаря механизму «подсматривания» и работе кэшей соседних ядер [20, 3447].
7. mem_load_uops_llc_hit_retired:xsnp_hitm — обработанные микрооперации загрузки, источники данных для которых были отображены в общий кэш последнего уровня, обслуженные по сигналу HitM [20, с. 3447].
8. mem_load_uops_llc_miss_retired — обработанные микрооперации загрузки, источники данных для которых не были отображены в кэш последнего уровня, увенчавшиеся кэш-промахом.
9. mem_load_uops_retired:l1_hit — обработанные микрооперации загрузки, источники данных для которых были отображены в кэш первого уровня, обслуженные с кэш-попаданием.
10. mem_load_uops_retired:l2_hit — обработанные микрооперации загрузки, источники данных для которых были отображены в кэш второго уровня, обслуженные с кэш-попаданием.
11. mem_load_uops_retired:llc_hit — обработанные микрооперации загрузки, источники данных для которых были отображены в кэш последнего уровня, обслуженные благодаря кэш-попаданию без механизма «подсматривания».
12. mem_uops_retired:all_loads — обработанные микрооперации загрузки.

Рассматриваются следующие величины, характеризующие частоту промахов и попаданий в кэш различного уровня [21]:

1. L1_hr = 100 * mem_load_uops_retired:l1_hit / mem_uops_retired:all_loads — доля попаданий в

кэш первого уровня, %.

2. L1_mr = 100 * (mem_uops_retired:all_loads - mem_load_uops_retired:l1_hit) / mem_uops_retired:all_loads — доля промахов для кэша первого уровня, %.
3. L2_hr = 100 * l2_rqsts:demand_data_rd_hit / l2_rqsts:all_demand_data_rd — доля попаданий в кэш второго уровня, %.
4. L2_mr = 100 * (l2_rqsts:all_demand_data_rd - l2_rqsts:demand_data_rd_hit) / l2_rqsts:all_demand_data_rd — доля промахов для кэша второго уровня, %.
5. L3_access = mem_load_uops_retired:llc_hit + mem_load_uops_llc_hit_retired:xsnp_hit + mem_load_uops_llc_hit_retired:xsnp_hitm + mem_load_uops_llc_miss_retired — обработанные микрооперации загрузки, зафиксированные на уровне кэша последнего уровня (учитываются и промахи, и попадания в кэш).
6. DD_L2_mr = 100 * L3_access / l2_rqsts:all_demand_data_rd — доля запросов в кэш второго уровня, увенчавшихся промахом (и повлекших запрос в кэш последнего уровня), %.
7. DD_L3_mr = 100 * mem_load_uops_llc_miss_retired / L3_access — доля запросов в кэш последнего уровня, увенчавшихся промахом, %.
8. CSS_L3_miss = (180 * mem_load_uops_llc_miss_retired) / CPU_CLK_UNHALTED — доля тактов процессора, затраченных на обслуживание промахов для кэша последнего уровня.
9. CSS_L2_miss = ((26 * mem_load_uops_retired:llc_hit) + (43 * mem_load_uops_llc_hit_retired:xsnp_hit) + (60 * mem_load_uops_llc_hit_retired:xsnp_hitm)) / CPU_CLK_UNHALTED — доля тактов процессора, затраченных на обслуживание промахов для кэша второго уровня (попаданий для кэша последнего уровня).
10. CSS_L1_miss = (12 * mem_load_uops_retired:l2_hit) / CPU_CLK_UNHALTED — доля тактов процессора, затраченных на обслуживание промахов для кэша первого уровня (попаданий для кэша второго уровня).
11. LLC_mr = 100 * LLC_MISSES / LLC_REFS — доля запросов данных, адресованных в кэш последнего уровня и увенчавшихся кэш-промахом, %.

B. Результаты профилирования

Сбор данных производился отдельно для запуска программы с 1 (таблица I) и 8 (таблица II) нитями OpenMP. Таблица I отличается наличием столбца vanilla, содержащего информацию о запуске исходного последовательного варианта atax_p. Некоторые значения, измеряемые в процентах, выходят за границы допустимого интервала из-за несовершенства семплирующего профилировщика, в особенности с многопоточным запуском приложения. Для значений аппаратных счетчиков событий и величин, характеризующих частоту промахов и попаданий в кэш различного уровня, рассчитан коэффициент корреляции Пирсона (столбец «Corr.») относительно времени выполнения программы (строка «Time spent, ms») по всем рассмотренным параллельным вариантам atax.

Характеристика LLC_mr показала отрицательную корреляцию со временем выполнения программы, в то время как LLC_MISSES и LLC_REFS, на основе которых она вычисляется, — положительную. Это свидетельствует о том, что сам по себе факт обращения в кэш последнего уровня является причиной снижения быстродействия, что подтверждается и L3_access. L1_mr и L2_mr показали положительную корреляцию со временем выполнения программы, при этом более сильная взаимосвязь наблюдается для однопоточных запусков. DD_L3_mr в зависимости от количества нитей показывает различный характер взаимосвязи: меняется знак коэффициента корреляции, при этом по модулю значения близки. DD_L2_mr, напротив, показывает положительную корреляцию для запусков в 1 и 8 нитей OpenMP. Набор характеристик CSS_L3_miss и CSS_L2_miss демонстрирует то же свойство. Для этих трех величин более сильная взаимосвязь наблюдается в случае однопоточных запусков. CSS_L1_miss показывает сильную взаимосвязь (положительную корреляцию) только для запусков в 8 нитей. Для однопоточных запусков не наблюдается явной линейной взаимосвязи.

V. Заключение

Среди рассмотренных характеристик частоты промахов и попаданий в кэш различного уровня показали неизменную (не зависящую от количества нитей OpenMP) взаимосвязь со временем выполнения программы те, что касаются кэша второго уровня, а также CSS_L3_miss, L3_access. В большинстве случаев наблюдается более сильная взаимосвязь со временем выполнения программы при однопоточных запусках как для значений счетчиков, так и для вычисляемых на их основе характеристик.

Рассмотренный пример программы atax 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. Критерии оптимальности аффинных отображений, применяемые в полиэдральной оптимизации, не всегда позволяют выбрать решения, ведущие к практически значимому ускорению вычислений при распараллеливании линейных программ, и потому требуют дальнейшего совершенствования. Например, комбинирование других методов нахождения аффинных расписаний вычислений

с известной жадной схемой Футриера может привести к значительно более эффективным параллельным вычислениям, чем одна только жадная схема Футриера, даже с модификациями для применения линейного целочисленного программирования. Одним из направлений развития методов нахождения аффинных отображений программ линейного класса может быть оптимизация кода для конкретной стратегии работы кэшей с сильной привязкой к целевой микроархитектуре.

Таблица I Результаты профилирования для запуска параллельных вариантов atax с 1 нитью OpenMP.

Features	ilp_sync	p_ilp_sync	p_ilp_sync_mdp	p_pluto	pluto	vanilla	Corr.
CPU_CLK_UNHALTED	2900913142	4971102042	2899131189	4922910160	2863426422	2479592387	0,9931
CSS_L1_miss	0,0104	0,0092	0,0109	0,0115	0,0107	0,0078	0,1495
CSS_L2_miss	0,0022	0,2289	0,0079	0,2302	0,0028	0,0025	0,9985
CSS_L3_miss	0,0089	0,2383	0,0178	0,2392	0,0085	0,0025	0,9984
DD_L2_mr	0,7696	32,0450	2,0335	31,7140	0,8419	0,9691	0,9983
DD_L3_mr	37,9083	13,0709	27,8271	13,0547	30,9209	13,0228	-0,6461
L1_hr	97,7275	91,0663	96,8068	92,1423	99,4098	99,5269	-0,9634
L1_mr	2,2725	8,9337	3,1932	7,8577	0,5902	0,4731	0,9634
L2_hr	90,6699	36,8608	85,5761	43,4040	89,6708	92,9144	-0,9935
L2_mr	9,3301	63,1392	14,4239	56,5960	10,3292	7,0856	0,9935
L3_access	379579	50345365	1030095	50116767	437371	266886	0,9988
LLC_MISSES	9168560	18704908	11127039	14781699	9109039	7016977	0,9198
LLC_REFS	14464632	109473152	19841269	100311209	15246266	9839429	0,9967
LLC_mr	63,3861	17,0863	56,0803	14,7358	59,7460	71,3149	-0,9789
l2_rqsts:all_demand_data_rd	49321548	157108159	50656994	158027412	51947850	27540978	0,9905
l2_rqsts:demand_data_rd_hit	44719805	57911348	43350284	68590232	46582045	25589530	0,8301
mem_load_uops_llc_hit_retired:xsnp_hit	1350	981	14740	1835	244	298	-0,2496
mem_load_uops_llc_hit_retired:xsnp_hitm	8244	5007	97172	10779	1793	1950	-0,2659
mem_load_uops_llc_miss_retired	143892	6580575	286646	6542583	135239	34756	0,9989
mem_load_uops_retired:l1_hit	942432004	987476171	940267831	932069531	921281544	729341784	0,4638
mem_load_uops_retired:l2_hit	2521132	3815748	2639591	4703861	2557003	1611384	0,9069
mem_load_uops_retired:llc_hit	226093	43758802	631537	43561570	300095	229882	0,9988
mem_uops_retired:all_loads	964347111	1084348947	971282476	1011554733	926751203	732808544	0,6752
Time spent, ms	141,185	762,421	113,372	740,707	105,277	97,246	

Таблица II Результаты профилирования для запуска параллельных вариантов atax с 8 нитями OpenMP.

Features	ilp_sync	p_ilp_sync	p_ilp_sync_mdp	p_pluto	pluto	Corr.
CPU_CLK_UNHALTED	6240269335	7769693078	4649601177	7700456884	5688079743	0,9998
CSS_L1_miss	0,0049	0,0301	0,0054	0,0300	0,0051	0,9038
CSS_L2_miss	0,0060	0,0350	0,0077	0,0337	0,0074	0,8939
CSS_L3_miss	0,0060	0,0879	0,0111	0,0809	0,0061	0,8879
DD_L2_mr	3,0364	9,2881	3,4967	8,8388	3,3483	0,8876
DD_L3_mr	15,1394	27,3355	20,4505	26,3155	13,3612	0,6835
L1_hr	100,5404	94,4346	99,5008	96,6649	101,3608	-0,7842
L1_mr	-0,5404	5,5654	0,4992	3,3351	-1,3608	0,7842
L2_hr	81,9039	70,9675	75,4384	74,4746	81,1460	-0,5106
L2_mr	18,0961	29,0325	24,5616	25,5254	18,8540	0,5106
L3_access	1364054	13881438	1403728	13150285	1448632	0,9087
LLC_MISSES	9463791	21978861	14097974	17743795	10028873	0,6810
LLC_REFS	21876319	59800892	24582338	52651827	22474466	0,8781
LLC_mr	43,2604	36,7534	57,3500	33,7002	44,6234	-0,9608
l2_rqsts:all_demand_data_rd	44923463	149453886	40143939	148778955	43264404	0,9216
l2_rqsts:demand_data_rd_hit	36794052	106063740	30283965	110802532	35107351	0,9314
mem_load_uops_llc_hit_retired:xsnp_hit	165399	160180	117288	94510	175127	-0,1265
mem_load_uops_llc_hit_retired:xsnp_hit	130494	215661	133771	173130	184781	0,6587

m						
mem_load_uops_llc_miss_retired	206509	3794559	287070	3460564	193554	0,9010
mem_load_uops_retired:l1_hit	1068600899	1089061791	995977072	1021822511	1057360408	0,5034
mem_load_uops_retired:l2_hit	2551258	19480733	2105788	19270761	2398805	0,9168
mem_load_uops_retired:llc_hit	861652	9711038	865599	9422081	895170	0,9098
mem_uops_retired:all_loads	1062857115	1153243989	1000974066	1057076981	1043164677	0,7916
Time spent, ms	109,248	172,868	46,131	171,952	88,653	

БИБЛИОГРАФИЯ

[1] Jubb C. Loop Optimizations in Modern C Compilers //Columbia University, New York, str. – 2014. – Т. 15.

[2] Wolf M. High-performance embedded computing: applications in cyber-physical systems and mobile computing. – Newnes, 2014.

[3] В.В. Воеводин, Вл.В. Воеводин «Параллельные вычисления» – СПб.: БХВ-Петербург, 2002. – 608 с.

[4] Feautrier P., Lengauer C. Polyhedron Model //Encyclopedia of parallel computing. – 2011. – Т. 1. – С. 1581-1592.

[5] Bondhugula U. et al. Automatic transformations for communication-minimized parallelization and locality optimization in the polyhedral model //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 Berlin Heidelberg, 2008. – С. 132-146.

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

[7] Sjödin J. et al. Design of graphite and the Polyhedral Compilation Package //GCC Developers' Summit. – 2009.

[8] Bondhugula U. et al. A practical automatic polyhedral parallelizer and locality optimizer //Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation. – 2008. – С. 101-113.

[9] Ахо Альфред В., Лам Моника С., Сети Рави, Ульман Джеффри Д. Компиляторы: принципы, технологии и инструментарий, 2-е изд.: Пер. с англ //М.: ООО «ИД Вильямс», 2008. – 1184 с.

[10] Лебедев А. С. Пространственно-временные преобразования при распараллеливании линейных программ //Информационные технологии и вычислительные системы. – 2015. – №. 1. – С. 19-32.

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

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

[13] Feautrier P. Some efficient solutions to the affine scheduling problem. Part II. Multidimensional time //International journal of parallel programming. – 1992. – Т. 21. – С. 389-420.

[14] Pouchet, L.-N. PolyBench/C 4.1 / L.-N. Pouchet, T. Yuki. — 2015. – – URL: https://sourceforge.net/projects/polybench/.

[15] Levon J., Elie P. Oprofile: A system profiler for linux. – 2004.

[16] Feautrier P. Parametric integer programming //RAIRO-Operations Research. – 1988. – Т. 22. – №. 3. – С. 243-268.

[17] Griebl M. Automatic parallelization of loop programs for distributed memory architectures. – Passau, Germany : Univ. Passau, 2004.

[18] Белоусов А. И., Ткачёв С. Б. Дискретная математика: учебник. 5-е изд. //М: Изд-во МГТУ им. Н.Э. Баумана. – 2015.

[19] Intel Ivy Bridge Microarchitecture events. URL: https://oprofile.sourceforge.io/docs/intel-ivybridge-events.php

[20] Intel® 64 and IA-32 Architectures Software Developer's Manual Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, and 4. – 2023. URL: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html

[21] How to measure L1, L2 and LLC cache miss rate with ocount (OProfile). Yunming Zhang's Blog. URL: https://yunmingzhang.wordpress.com/2015/07/09/how-to-measure-l1-l2-and-llc-cache-miss-rate-with-ocount-oprofile/

Лебедев Артем Сергеевич, старший преподаватель МИРЭА – Российский технологический университет (https://www.mirea.ru/). E-mail: tementy@gmail.com
E-library: AuthorID 723581
ORCID: 0000-0002-8940-8269

The impact of polyhedral optimizations on data locality when parallelizing affine programs in the context of ATAX case study

Artem S. Lebedev

***Abstract*—The article examines the impact of polyhedral optimization on an important property of the program — locality of data use, which is closely related to the possible limitation of its performance both in sequential and parallel execution. A new method for finding multidimensional schedules of affine programs is proposed, combining topological sorting of the vertices of a generalized dependence graph and the greedy scheme developed by P. Feautrier (with modifications previously proposed by the author). Experimental results of studying the performance of parallel versions of the polybench/atax program obtained using the modern translator (pluto) and the translator developed by the author (ilpy) are presented, which allows to compare two approaches to setting optimization problems when finding affine mappings: lexicographic optimization (pluto) and integer linear programming (ilpy). Using the oprofile/ocount profiling tools, the inefficiency of using multi-core processor caches was revealed and the need to improve the criteria for the optimality of schedules and placements of computations for affine programs was shown.**

***Keywords*—affine programs, data locality, polyhedral model, profiling.**

References

[1] Jubb, C. (2014). Loop Optimizations in Modern C Compilers. Columbia University, New York, str, 15.

[2] Wolf, M. (2014). High-performance embedded computing: applications in cyber-physical systems and mobile computing. Newnes.

[3] Voevodin V.V., and Voevodin Vl.V. Parallel computations. BHV, Saint-Petersburg (2002). (In Russ.)

[4] Feautrier, P., and Lengauer, C. (2011). Polyhedron Model. Encyclopedia of parallel computing, 1, 1581-1592.

[5] Bondhugula, U., Baskaran, M., Krishnamoorthy, S., Ramanujam, J., Rountev, A., and Sadayappan, P. (2008). Automatic transformations for communication-minimized parallelization and locality optimization in the polyhedral model. In 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 (pp. 132-146). Springer Berlin Heidelberg.

[6] Grosser, T., Groesslinger, A., and Lengauer, C. (2012). Polly—performing polyhedral optimizations on a low-level intermediate representation. Parallel Processing Letters, 22(04), 1250010.

[7] Sjödin, J., Pop, S., Jagasia, H., Grosser, T., and Pop, A. (2009, June). Design of graphite and the Polyhedral Compilation Package. In GCC Developers' Summit.

[8] Bondhugula, U., Hartono, A., Ramanujam, J., and Sadayappan, P. (2008, June). A practical automatic polyhedral parallelizer and locality optimizer. In Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation (pp. 101-113).

[9] Alfred Aho, V., Monica Lam, S., Ravi, S., and Jeffrey Ullman, D. Compilers Principles, Techniques, 2nd ed. Translation from English. ID Williams LLC, Moscow (2008). (In Russ.)

[10] A. S. Lebedev. (2015). Space-time mappings for parallelization of affine programs. Informatsionnyye tekhnologii i vychislitel'nyye sistemy, 1, 19–32. (In Russ.)

[11] A. S. Lebedev, and V. I. Solodovnikov. (2023). Translation of affine programs for parallel execution on universal multi-core processors. Proceedings of the Institute for Systems Analysis Russian Academy of Sciences, 73(4), 36-47. (In Russ.)

[12] Feautrier, P. (1992). Some efficient solutions to the affine scheduling problem. I. One-dimensional time. International journal of parallel programming, 21, 313-347.

[13] Feautrier, P. (1992). Some efficient solutions to the affine scheduling problem. Part II. Multidimensional time. International journal of parallel programming, 21, 389-420.

[14] Pouchet, L. N., & Yuki, T. (2015). PolyBench/C 4.1. SourceForge. Available online: https://sourceforge.net/projects/polybench/.

[15] Levon, J., & Elie, P. (2004). Oprofile: A system profiler for linux.

[16] Feautrier, P. (1988). Parametric integer programming. RAIRO-Operations Research, 22(3), 243-268.

[17] Griebl, M. (2004). Automatic parallelization of loop programs for distributed memory architectures. Passau, Germany: Univ. Passau.

[18] A. I. Belousov, and S. B. Tkachev. Discrete mathematics: textbook, 5th ed. Moscow, Publishing house of Bauman Moscow State Technical University (2015). (In Russ.)

[19] Intel Ivy Bridge Microarchitecture events. Available online: https://oprofile.sourceforge.io/docs/intel-ivybridge-events.php

[20] Intel® 64 and IA-32 Architectures Software Developer's Manual Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, and 4. Order Number: 325462-082US. December 2023. Available online: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html

[21] How to measure L1, L2 and LLC cache miss rate with ocount (OProfile). Yunming Zhang's Blog. Available online: https://yunmingzhang.wordpress.com/2015/07/09/how-to-measure-l1-l2-and-llc-cache-miss-rate-with-ocount-oprofile/

Artem Lebedev, lecturer at MIREA – Russian Technological University (https://www.mirea.ru/).
E-mail: tementy@gmail.com
E-library: AuthorID 723581
ORCID: 0000-0002-8940-8269