Трансляция программ линейного класса для параллельного выполнения на универсальных многоядерных процессорах

А.С. Лебедев, В.И. Солодовников

Федеральное государственное бюджетное учреждение науки «Центр информационных технологий в проектировании Российской академии наук», МО, Россия

Аннотация. Рассматривается задача нахождения пространственных и временных отображений программ линейного класса, обеспечивающих локальность использования данных, при распараллеливании гнезд циклов для выполнения на универсальных многоядерных процессорах. Предложена архитектура транслятора текст-в-текст для выполнения распараллеливающих преобразований линейных программ с поддержкой технологии OpenMP. Приведены экспериментальные результаты исследования производительности распараллеленных программ из набора тестов polybench: lu, syr2k, gramschmidt. Произведено сравнение производительности параллельных программ, полученных применением разработанного транслятора ilpy и современного транслятора pluto 0.11.4.

Ключевые слова: *линейная программа, модель многогранников, пространственные и временные отображения, транслятор.*

DOI: 10.14357/20790279230405 **EDN: WNRRTT**

Введение

Основная идея распараллеливания циклов основывается на том, что независимые итерации цикла могут быть выполнены параллельно. Классические методы распараллеливания модифицируют текстовое представление структуры циклов в программе до состояния, в котором некоторые циклы становятся параллельными. Этот подход называется распараллеливанием на основе текста и подразумевает применение следующих преобразований: расщепление циклов, слияние циклов, переиндексирование, масштабирование, реверс индекса цикла, перестановка порядка циклов, сдвиг циклов [1]. Сложность практического применения подхода состоит в правильном выборе набора этих преобразований и порядка их применения. Наиболее успешным отечественным проектом, базирующимся на применении таких преобразований, является Открытая распараллеливающая система [2].

Распараллеливание на основе модели – другой подход, игнорирующий особенности текстового представления программы, и использующий преобразования в рамках определенной математической модели. Если модель обладает достаточной степенью общности, то ожидаемое преимущество по сравнению с распараллеливанием на основе текста заключается в том, что корректная последовательность шагов распараллеливания может быть представлена одним преобразованием, и это преобразование может быть построено с помощью классических математических методов оптимизации [3]. Оригинальный фреймворк для автоматического распараллеливания плотно вложенных циклов, в дальнейшем названный моделью многогранников (polytope model), был предложен К. Ленгауэром [4]. Современное состояние модели многогранников допускает неплотно вложенные циклы, а также аффинные зависимости. В настоящей работе модель многогранников рассматривается как модель последовательных и параллельных вычислений, применимая для распараллеливания программ, принадлежащих линейному классу [5, с. 340].

Первая задача в этом подходе, основанном на модели, – определить для каждого вычисления, когда оно должно выполняться. Полученное временное распределение называется расписанием вычислений. Параллелизм выражается в исполнении множества вычислительных операций в рам-

ках одного временного шага. Ограничениями для такого планирования вычислений являются информационные зависимости между операциями. Кроме расписания существует также пространственное распределение – размещение вычислений, определяющее для каждого вычисления, где оно должно происходить. В настоящей работе рассматривается реализация метода нахождения расписания и размещения вычислений, разработанного автором, и доведенного до практического воплощения в трансляторе текст-в-текст [6,7].

1. Этапы распараллеливания программ в модели многогранников

Созданное программное обеспечение транслирует исходную последовательную программу на языке С в ее оптимизированный для параллельного выполнения вариант на том же языке. Дальнейшие отсылки к разработанному транслятору будут включать его название ilpy (от аббревиатуры ILP – Integer Linear Programming). Все этапы распараллеливания программы с применением ilpy включены в диаграмму потоков данных на рис. 1. Распараллеливающие преобразования реализуются транслятором ilpy с применением сторонних библиотек и их модификаций: osl [8] – представление полиэдрального представления программы OpenSCOP в памяти и взаимодействия с библиотеками clan, candl, cloog; clan [9] – извлечение полиэдрального представления программы; candl [10] – нахождение многогранников зависимостей; cloog [11] используется в модифицированном виде для генерации параллельных программ на языке С в соответствии с найденными аффинными отображениями; polylib [12] используется для вычисления количества точек с целочисленными координатами внутри многогранника; glpk [13] – решатель задач линейного целочисленного программирования; openblas [14] – матричная арифметика. Постобработка полученного параллельного кода включает преобразования, выполняемые машинным способом – расстановка директив OpenMP. Финальная компиляция выполняется с помощью компилятора gcc.

2. Полиэдральное представление программы OpenSCOP

OpenSCOP [8] – это открытая спецификация, которая определяет формат файла и набор структур данных для представления кода со статическим потом управления в модели многогранников. Ключевой абстракцией OpenSCOP является отношение (relation), с помощью которой может быть представлен многогранник. Многогранник может описывать домен инструкции, ограничения на внешние переменные программы (контекст), доступ к массиву на чтение и запись, зависимость по данным, аффинное отображение. Примеры отношений для программы LU-разложения квадратной матрицы A из пакета polybench [15] приведены на рис. 2.

Фрагменты текста, начинающиеся с символа «#», являются комментариями. Многогранник описывается матрицей ограничений. Ограничение представляет равенство $p(x)=0$, если первый элемент строки содержит 0, или неравенство $p(x)\geq 0$, если первый элемент строки содержит 1. Следующие элементы являются коэффициента-

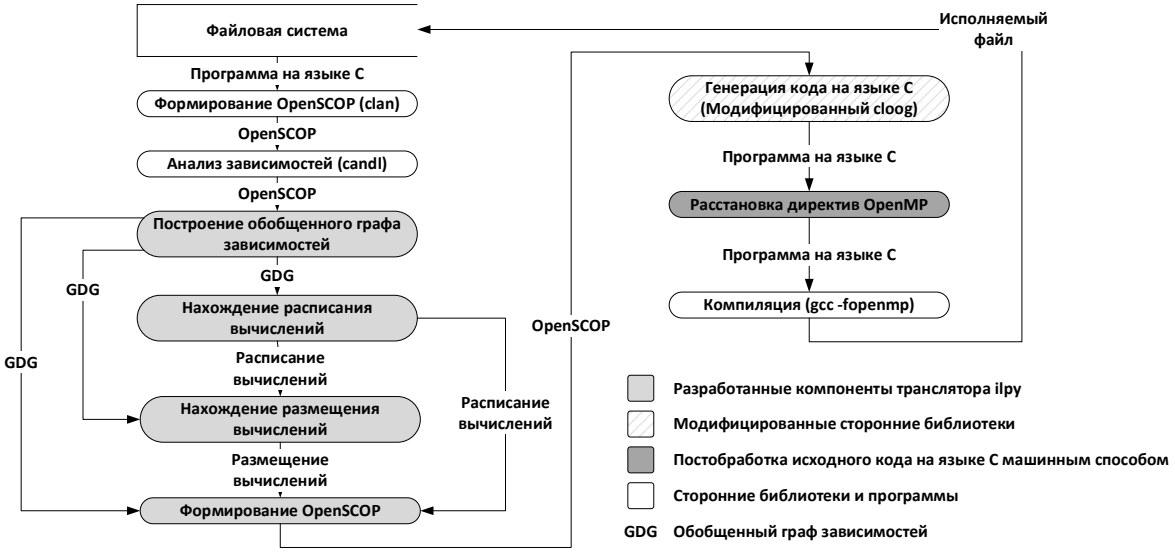
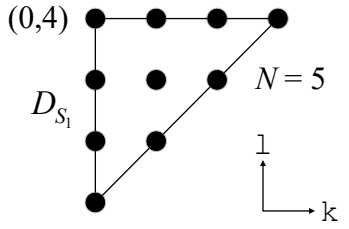


Рис. 1. Распараллеливание линейной программы с применением ilpy

```
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
}
```



Аффинные отображения для S_1:

```
SCATTERING

5 10 5 2 0 1

# e/i| c1   c2   c3   c4   c5 |  k    l |  N |  1

   0   -1    0    0    0    0    0    0    0    0 ## c1==0

   0    0   -1    0    0    0    1    0    0    0 ## c2==k

   0    0    0   -1    0    0    0    0    0    0 ## c3==0

   0    0    0    0   -1    0    0    1    0    0 ## c4==l

   0    0    0    0    0   -1    0    0    0    0 ## c5==0
```

Домен D_{S_1}:

```
DOMAIN

6 5 2 0 0 1

# e/i|  k    l |  N |  1

   1    1    0    0    0 ## k >= 0

   1   -1    0    1   -1 ## -k+N-1 >= 0

   1    0    0    1   -1 ## N-1 >= 0

   1   -1    1    0   -1 ## -k+l-1 >= 0

   1    0   -1    1   -1 ## -l+N-1 >= 0

   1   -1    0    1   -2 ## -k+N-2 >= 0
```

Рис. 2. Многогранники в OpenSCOP

ми выходных измерений, за которыми следуют коэффициенты входных измерений, затем локальных измерений, и, наконец, параметров. Последний элемент является постоянным членом.

3. Анализ зависимостей по данным и обработка многогранников зависимостей

Расширение dependence спецификации OpenSCOP позволяет представлять многогранники зависимостей отношениями общего вида. В табл. 1 представлена структура матрицы ограничений такого отношения для некоторого ребра e в обобщенном графе зависимостей. Строки матрицы ограничений сгруппированы в шесть секций: домен начальной вершины $\sigma(e)$; домен конечной вершины $\delta(e)$; индексная функция доступа в $\sigma(e)$; индексная функция доступа в $\delta(e)$; $1+p_A$ уравнений, фиксирующих совпадение адресов ячейки памяти, к которой осуществляется доступ в $\sigma(e)$ и $\delta(e)$ (одно уравнение для номера массива A, остальные p_A – для его индексов); p_e уравнений и одно неравенство для описания лексикографического предшествования зависимых операций. Участие переменных в уравнениях и неравенствах отмечено знаком «+» в ячейках таблицы, сигнализирующим о возможном присутствии соответствующей переменной с ненулевым коэффициентом в уравнении или неравенстве. Заранее известные коэффициенты вписаны явно в виде скаляров и подматриц, а пустые ячейки таблицы соответствуют заполнению участков матрицы ограничений нулями.

Инструмент candl [10] генерирует отдельное описание зависимости для каждого возможного значения глубины, поэтому результат его работы может содержать различные описания для совпадающих инструкций и доступов к памяти, отличающиеся только указанной глубиной зависимости и последней секцией в матрице ограничений, фиксирующей лексикографический порядок зависимых операций. В таком описании отсутствует явное представление h-преобразования, что затрудняет реализацию техники «lastwriter», которая заключается в удалении транзитивных зависимостей путем вычисления последнего конфликтующего доступа в случаях RAW/WAW. Исключение из рассмотрения таких зависимостей, которые присутствуют в транзитивном замыкании Γ, но отсутствуют в Γ, не нарушает выполнения принципа причинности, но упрощает анализ программы. Кроме того, устранение зависимостей типа WAR/WAW дает больше потенциальных возможностей для парал-

Табл. 1

Представление многогранника зависимостей в OpenSCOP

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

лельного выполнения операций [16, с. 15]. Транслятор pluto использует библиотеку isl [17] вместо candl, если пользователь требует включения опции --lastwriter. В ilpy подобное поведение достигается предобработкой зависимостей, а именно секции лексикографического предшествования в матрице ограничений: если секция не пуста, и последнее условие является неравенством $p(x) \geq 0$, то оно заменяется на равенство $p(x) = 0$. Технически в последней строке матрицы ограничений значение в первом столбце изменяется с 1 на 0. Если многогранник зависимости остался непустым, значит он определяет последний конфликтующий доступ согласно лексикографическому порядку, и модификация корректна. В противном случае модификация некорректна, так как влечет потерю информации о зависимости.

4. Формирование ограничений для зависимостей по данным

При нахождении расписаний и размещений вычислений [6] фигурируют следующие условия, порождаемые каждой зависимостью по данным *e*:

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

Каждое из этих условий преобразуется применением леммы Фаркаша, а затем метода неопределенных коэффициентов. Для того, чтобы применить лемму Фаркаша, необходимо обеспечить в описании многогранников только условия в виде неравенств. Это можно сделать заменой каждого условия в виде равенства $p(x) = 0$ на два неравенства: $p(x) \geq 0$ и следующего за ним $-p(x) \geq 0$. Матрица ограничений каждого многогранника зависимостей подвергается указанной трансформации, при этом первый столбец отбрасывается, так как все условия становятся единообразны. Уд-

ваивается количество условий в секциях 3, 4, 5 и 6 (последнее неравенство, если оно присутствовало, было заменено равенством на этапе предобработки). Порядок следования условий сохраняется. Для удобства дальнейшего изложения введем обозначения для индексов первого и последнего условия в каждой секции матрицы ограничений многогранника зависимости: $\text{source}_{\text{domain}}^{\text{start}},\ldots,\text{source}_{\text{domain}}^{\text{end}}$; $\text{target}_{\text{domain}}^{\text{start}},\ldots,\text{target}_{\text{domain}}^{\text{end}}$; $\text{source}_{\text{access}}^{\text{start}},\ldots,\text{source}_{\text{access}}^{\text{end}}$; $\text{target}_{\text{access}}^{\text{start}},\ldots,\text{target}_{\text{access}}^{\text{end}}$; $\text{access}_{\text{eq}}^{\text{start}},\ldots,\text{access}_{\text{eq}}^{\text{end}}$; $\text{precedence}^{\text{start}},\ldots,\text{precedence}^{\text{end}}$. Индексирование начинается с нуля, то есть $\text{source}_{\text{domain}}^{\text{start}}=0$. Поскольку секции следуют друг за другом без зазоров в матрице ограничений, будем использовать введенные обозначения индексов для указания фрагментов матрицы, захватывающих более одной секции.

Обозначим m_D матрицу ограничений, описывающую многогранник D, $\mu_S^{sign}\in\{-1,1\}$ – знак аффинного отображения φ_S инструкции S в рассматриваемом условии. Условие DEP_UPPER_BOUND подразумевает $\mu_{\sigma(e)}^{sign}=1$, $\mu_{\delta(e)}^{sign}=-1$. Для всех остальных условий знаки обратные. Приравнивание коэффициентов при индексах итерации, индексах массива, внешних переменных программы, а также констант, порождает новые условия в виде равенств для каждой зависимости по данным e. Зависимости по данным, как и инструкции, перебираются в порядке возрастания их номеров (определяется внутренними механизмами библиотеки osl), и для каждой из них формируются 6 групп ограничений в виде равенств. Новые ограничения представляются в виде строк матрицы, имеющей структуру, представленную в табл. 2.

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

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

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

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

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

Табл. 2

Структура матрицы для хранения ограничений в виде равенств

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

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

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

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

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

$$- \sum_{k=\text{precedence}^{start}}^{\text{precedence}^{end}} \lambda^*_{e,k+1} m_{R_e}[k]\left[v-1+p_{\sigma(e)}+(p_A+1)\right] = 0.$$

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

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

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

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

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

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

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

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

6. Для констант:

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

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

$$-\lambda^*_{e,0} - \sum_{k=\text{source}^{start}_{domain}}^{\text{target}^{end}_{access}} \lambda^*_{e,k+1} m_{R_e}[k]\left[p_{\sigma(e)}+2(p_A+1)+p_{\delta(e)}+q_z\right] -$$

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

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

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

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

5. Нахождение расписания вычислений

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

1. Выяснение зависимостей, которые можно удовлетворить очередным компонентом многомерного расписания. Конструируются ограничения вида LEGALITY_DECISION. Для индикаторных переменных ε_e не конструируются отдельные ограничения, выполняется только вызов glp_set_col_bnds(…, GLP_DB, 0, 1) для фиксации $\varepsilon_e \in \{0,1\}$.
2. Вычисление очередного компонента многомерного расписания. Конструируются ограничения вида LEGAL_SHEDULE и DEP_UPPER_BOUND.

Для каждого ограничения в виде равенства выполняется вызов glp_set_row_bnds(…, GLP_FX, fv, fv), где fv представляет константу в правой части ограничения. Для всех столбцов, кроме соответствующих индикаторным переменным, фиксируется неотрицательность: glp_set_row_bnds(…, GLP_LO, 0, 0). Коэффициенты при переменных в целевой функции задаются вызовом glp_set_obj_coef, при этом вместо $\vec{z}$ используются веса внешних переменных программы, которые можно передавать в параметрах командной строки.

Решение задачи ЛЦП считается успешным при выполнении следующих условий:

1. glp_simplex(lp, NULL) == 0 – решатель задачи линейного программирования (ЛП) завершил работу успешно;
2. glp_get_status(lp) == GLP_OPT – решатель задачи ЛП нашел оптимум;
3. glp_intopt(lp, NULL) == 0 – решатель задачи линейного целочисленного программирования (ЛЦП) завершил работу успешно;
4. glp_mip_status(lp) == GLP_OPT – решатель задачи ЛЦП нашел оптимум.

В противном случае транслятор прерывает работу.

6. Нахождение размещения вычислений

Конструируются ограничения вида FCO_PLACEMENT и DEP_UPPER_BOUND если

фиксируется свойство вперед направленных коммуникаций [18]. В противном случае применяется DEP_LOWER_BOUND вместо FCO_PLACEMENT.

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

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

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

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

$$-b+1 \le \sum_{k=1}^{p_{D_{S_i}}} \mu_{S_i,k} m_{\vec{x}_{S_i}}[j-1][k-1] \le b-1, \quad j = 1,\ldots,p;$$

$$\sum_{k=1}^{p_{D_{S_i}}} \left(\mu_{S_i,k} \sum_{j=1}^{p} b^{j-1} m_{\vec{x}_{S_i}}[j-1][k-1] \right) + \varepsilon b^{p+1} \ge 1;$$

$$-\sum_{k=1}^{p_{D_{S_i}}} \left(\mu_{S_i,k} \sum_{j=1}^{p} b^{j-1} m_{\vec{x}_{S_i}}[j-1][k-1] \right) - \varepsilon b^{p+1} \ge 1 - b^{p+1}.$$

Значение параметра b выбрано равным 5 согласно рекомендациям [19]. В левой части второго и третьего ограничения-неравенства скалярное произведение раскрыто для $\vec{y} : \vec{y}^{(j)} = b^{j-1}, \quad j = 1,\ldots,p$.

Для каждого ограничения-неравенства добавляется строка в матрицу ограничений задачи ЛЦП. Значение элемента *j* этой строки определяется коэффициентом при переменной в левой части неравенства, указанной в названии столбца *j* матрицы ограничений (табл. 2). Для второго и третьего ограничения-неравенства матрица ограничений задачи ЛЦП дополняется столбцом для индикаторной переменной $\varepsilon \in \{0,1\}$. Чтобы задать семантику неравенства для строки матрицы ограничений задачи ЛЦП, для первого двойного неравенства может быть использован вызов glp_set_row_bnds(…, GLP_DB, -b + 1, b - 1), для второго – glp_set_row_bnds(…, GLP_LO, 1, 0)}, для третьего – glp_set_row_bnds(…, GLP_LO, 1 - bp, 0), где bp вычислено как b^{p+1}.

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

Рассмотрим скалярное произведение для $\vec{y} : \vec{y}^T = \mathbf{H}_{S_i}^{\perp}[r], \quad r \in [1..p_{S_i}]$ и $\vec{x}_{S_i} = \vec{v}_{S_i}$:

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

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

Пусть $\vec{x}_{S_i} = \mathbf{H}_{S_i}^{\perp} \vec{v}_{S_i}$, тогда матрица коэффициентов задается следующим образом:

$$m_{\vec{x}_{S_i}}[r-1][k-1] = \sum_{j=1}^{p_{S_i}} \mathbf{H}_{S_i}^{\perp}[r][j] m_{D_{S_i}}[k-1][j],$$

$$r = 1,\ldots,p_{S_i}, k = 1,\ldots,p_{D_{S_i}}.$$

Конструирование ограничений-неравенств для фиксации линейной независимости размещения вычислений от компонентов ранее найденного многомерного расписания сводится к конструированию ограничений для $\vec{x}_{S_i} \neq \vec{0}$.

Для каждого ограничения в виде равенства выполняется вызов glp_set_row_bnds(…, GLP_FX, fv, fv), где fv представляет константу в правой части ограничения. Для всех столбцов, кроме соответствующих индикаторным переменным, по умолчанию фиксируется неотрицательность: glp_set_row_bnds(…, GLP_LO, 0, 0).

Решение задачи ЛЦП считается успешным при выполнении следующих условий:

1. glp_simplex(lp, NULL) == 0 – решатель задачи ЛП завершил работу успешно;
2. glp_get_status(lp) == GLP_OPT – решатель задачи ЛП нашел оптимум;
3. glp_intopt(lp, NULL) == 0 или glp_intopt(lp, NULL) == GLP_ETMLIM – решатель задачи ЛЦП завершил работу успешно или прервал вычисления по достижению заданного пользователем лимита времени;
4. glp_mip_status(lp) == GLP_OPT или glp_mip_status(lp) == GLP_FEAS – решатель задачи ЛЦП нашел оптимум или остановился на допустимом субоптимальном решении.

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

При построении матрицы $\mathbf{H}_{S_i}, \quad i = 1,\ldots,m$ учитываются не только компоненты многомерного расписания, но и аффинные отображения, соответствующие размещению вычислений. Это

позволяет генерировать многомерные аффинные размещения вычислений подобно тому, как выполняется построение набора линейно независимых отображений в pluto [20], что легло в основу реализации опции --async. При этом в матрицу $\mathbf{H}_{S_i}$ не добавляются строки, полностью состоящие из нулей, и строки, совпадающие с предыдущими с точностью до постоянного множителя, поскольку они препятствуют вычислению $\mathbf{H}_{S_i}^{\perp}$. Если не удалось сформировать ни одной строки, то $\mathbf{H}_{S_i}^{\perp} = I_{p_{S_i}}$, что означает допустимость использования любых выходных измерений D_{S_i} для вычисления компонента многомерного размещения вычислений. Если $\mathbf{H}_{S_i}^{\perp}$ содержит только нули, то ограничение линейной независимости не накладывается в силу невозможности его выполнения.

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

$d \leftarrow 1$.

1. Вычислить $\pi_{S_i}^{(d)}, i = 1,\ldots,m$ как оптимальный набор аффинных отображений для E.
2. Если требуется одномерное размещение вычислений, то вернуться к Шагу 1.
3. Если существует инструкция S, для которой найдено менее p_S аффинных отображений (с учетом компонентов многомерного расписания и многомерного размещения вычислений), то выполнить $d \leftarrow d+1$ и перейти к Шагу 2.
4. Иначе вернуть d.

Результатом работы алгоритма является размещение вычислений, а также количество найденных аффинных отображений.

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

- линейная независимость: ДА, свойство FCO: ДА;
- линейная независимость: ДА, свойство FCO: НЕТ;
- линейная независимость: НЕТ, свойство FCO: ДА;
- линейная независимость: НЕТ, свойство FCO: НЕТ.

7. Генерация программного кода

Найденные расписания и размещения вычислений фигурируют в представлении программы OpenSCOP в отношениях типа SCATTERING. Для каждой инструкции S исходное отношение заменяется на новое, генерируемое по схеме, проиллюстрированной в табл. 3.

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

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

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

Весь код, сгенерированный cloog, помещается в параллельный регион OpenMP с помощью директивы #pragma omp parallel. Параллельные циклы, имеющие только одну итерацию, помечаются директивой #pragma omp master, направляющей вычисления в главную нить без барьерной синхронизации. Параллельные циклы, имеющие более одной итерации, помечаются сразу двумя директивами. Во-первых, устанавливается явная барьерная синхронизация #pragma omp barrier, чтобы синхронизировать главную нить с остальными перед выполнением параллельного цикла. Во-вторых, отмечается распараллеливание цикла по нитям внутри параллельного региона с помощью #pragma omp for. Неявная барьерная синхронизация устанавливается после выполнения параллельного цикла. Таким образом, синхронизационные барьеры будут обрамлять только параллельные циклы с количеством итераций больше одной. В заголовки циклов добавляется тип счетчика int для соответствия стандарту C99.

Табл. 3

Структура матрицы для хранения аффинных отображений

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

8. Экспериментальные исследования производительности параллельных программ

Вызов транслятора pluto выполняется с параметрами --parallelize --lastwriter, что подразумевает распараллеливание на OpenMP и рассмотрение при анализе зависимостей только последней операции, индуцирующей зависимость по данным для двух инструкций, что соответствует поведению ilpy. Итоговая компиляция выполняется с помощью gcc с наивысшим уровнем оптимизации. С целью обеспечить идентичные условия при сравнении производительности параллельных программ, полученных применением ilpy и pluto, выполняется постобработка результатов работы последнего: конструкция omp parallel по возможности выносится из циклов и ветвлений так, чтоб не нарушить корректность программы. Все тестовые запуски выполнялись на оборудовании РТУ МИРЭА. Использовалась машина на базе процессора Intel(R) Xeon(R) CPU E5-2690 v2 @ 3.00GHz, имеющая 16Гб оперативной памяти стандарта DDR3. Операционная система – Linux CentOS 7.8 x64. Компилятор – gcc версии 4.8.5. На рис. 3 приведены результаты запуска параллельных вариантов программ lu, syr2k, gramschmidt. Вычисления производятся с двойной точностью (double) при заданных размерах задачи: lu (N=3072), syr2k (N=1536, M=1536), gramschmidt (M=2048, N=1024).

Для всех тестовых программ удалось получить ускорение вычислений как при распараллеливании с применением ilpy, так и при распараллеливании с применением pluto. При трансляции syr2k с помощью ilpy использовался параметр --async для получения более производительного решения. При трансляции gramschmidt была задана верхняя граница для множителей Фаркаша и коэффициентов ограничивающей аффинной формы L, равная 2, и дополнительно ограничено время работы решателя glpk тридцатью секундами, чтобы завершить нахождение размещения вычислений за при-

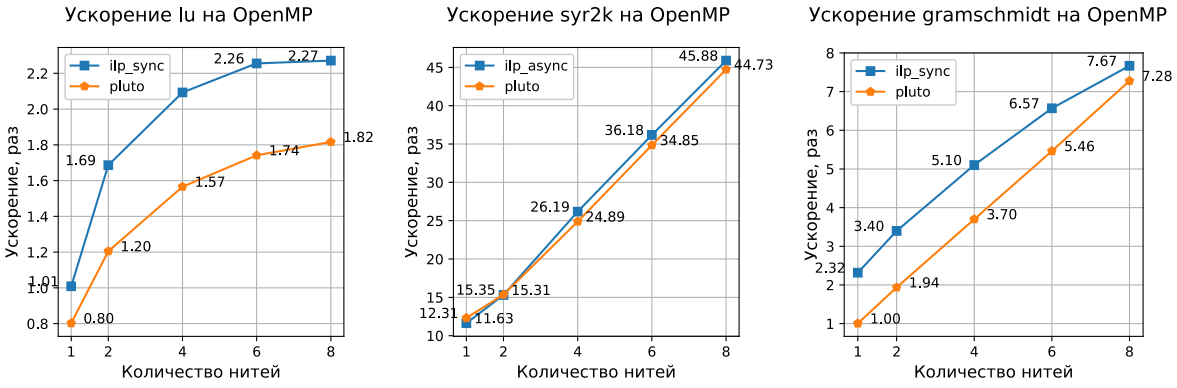


Рис. 3. Результаты запусков параллельных программ

емлемое время. Найденное решение принято как субоптимальное. Во всех запусках ilpy демонстрирует лучшую эффективность найденных решений благодаря разработанному методу нахождения пространственных и временных отображений [6]. Для восьми нитей выигрыш в эффективности составил 25.05% для lu, 2.59% для syr2k, 5.37% для gramschmidt.

Заключение

Результаты экспериментов показывают, что самыми удачными для распараллеливания оказались программы syr2k и gramschmidt, поскольку для них удалось достичь наибольшего прироста в производительности. Пример syr2k показал сверхлинейное ускорение за счет улучшения локальности использования данных при вычислениях: ускорение более 10 раз достигается даже при однопоточном запуске. Разработанное программное обеспечение – транслятор ilpy – может быть использовано для анализа потока данных в программах линейного класса, улучшения локальности использования данных при вычислениях, а также распараллеливания таких программ для универсальных многоядерных процессоров благодаря поддержке OpenMP в качестве технологии распараллеливания. В качестве направлений развития транслятора отмечаются решения следующих технических задач:

- поддержка отношений в виде объединения выпуклых многогранников в OpenSCOP;
- поддержка локальных измерений в многогранниках, описывающих домены, зависимости, доступы к данным, аффинные отображения;
- вычисление и применение h-преобразования для упрощения многогранников зависимостей;
- оптимизация ветвлений машинным способом как расширение функциональности cloog.

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

Литература

1. *Banerjee U.* Loop transformations for restructuring compilers: the foundations. – Springer Science & Business Media. 2007.
2. *Штейнберг Б.Я. и др.* Состояние и возможности Открытой распараллеливающей системы (лето 2006 г.) // Новосибирск: Академгородок. 2006. С. 28-29.
3. *Griebl M.* Automatic parallelization of loop programs for distributed memory architectures. – Passau, Germany : Univ. Passau. 2004.
4. *Lengauer C.* Loop parallelization in the polytope model // International Conference on Concurrency Theory. – Berlin, Heidelberg : Springer Berlin Heidelberg. 1993. P. 398-416.
5. *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления. СПб.: БХВ-Петербург. 2002. Т. 1. С. 608.
6. *Лебедев А.С.* Пространственно-временные преобразования при распараллеливании линейных программ //Информационные технологии и вычислительные системы. 2015. №. 1. С. 19-32.
7. Свидетельство о гос. регистрации программы для ЭВМ. Программа для построения аффинных преобразований программ линейного класса [Текст] : 2022667001 Рос. Федерация / А. С. Лебедев, С. В. И. ; Ф. государственное бюджетное учреждение науки Центр информационных технологий в проектировании Российской академии наук. № 2022666032 ; заявл. 29.08.2022 ; опубл. 13.09.2022.
8. *Bastoul C.* Openscop: A specification and a library for data exchange in polyhedral compilation tools //Paris-Sud University, France, Tech. Rep. 2011. T. 9. P. 22.
9. *Bastoul C.* Extracting polyhedral representation from high level languages //Tech. rep. Related to the Clan tool. LRI, Paris-Sud University. 2008.
10. *Bastoul C., Pouchet L.N.* Candl: the chunky analyzer for dependences in loops. – tech. rep., LRI, Paris-Sud University, France. 2012. С. 2.
11. *Bastoul C.* Code generation in the polyhedral model is easier than you think //Proceedings. 13th International Conference on Parallel Architecture and Compilation Techniques. 2004. PACT 2004. IEEE. 2004. С. 7-16.
12. *Loechner V.* PolyLib: A library for manipulating parameterized polyhedra. 1999.
13. *Makhorin A.* GLPK (GNU linear programming kit). URL: http://www.gnu.org/s/glpk/glpk.html (доступ 18 августа 2023).
14. OpenBLAS: an optimized BLAS library. URL: https://www.openblas.net/ (доступ 18 августа 2023).
15. *Yuki T, Pouchet L.* PolyBench 4.2. URL: https://sourceforge.net/projects/polybench/ (доступ 18 августа 2023).
16. *Лиходед Н.А.* Методы распараллеливания гнезд циклов. 2008.
17. *Verdoolaege S.* isl: An integer set library for the polyhedral model //International Congress on Mathematical Software. – Berlin, Heidelberg : Springer Berlin Heidelberg, 2010. P. 299-302.

18. *Griebl M., Feautrier P., Größlinger A.* Forward communication only placements and their use for parallel program construction //Languages and Compilers for Parallel Computing: 15th Workshop, LCPC 2002, College Park, MD, USA, July 25-27, 2002. Revised Papers 15. – Springer Berlin Heidelberg. 2005. P. 16-30.
19. *Bondhugula U., Cohen A.* Handling Negative Coefficients in Automatic Transformation Schedules. 2014.
20. *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. P. 132-146.

Лебедев Артем Сергеевич. Федеральное государственное бюджетное учреждение науки «Центр информационных технологий в проектировании Российской академии наук», Московская область, Одинцово, Россия. Младший научный сотрудник. Область научных интересов: параллельное программирование, обработка больших данных, оптимизирующие преобразования программ. e-mail: tementy@gmail.com (Ответственный за переписку).

Солодовников Владимир Игоревич. Федеральное государственное бюджетное учреждение науки «Центр информационных технологий в проектировании Российской академии наук», Московская область, Одинцово, Россия. Директор. Кандидат технических наук. Область научных интересов: методы искусственного интеллекта, самоорганизующиеся системы поддержки принятия решений, нейросетевой анализ данных. E-mail: info@ditc.ras.ru.

Translation of affine programs for parallel execution on universal multi-core processors

A.S. Lebedev, V.I. Solodovnikov

Center of Information Technologies in Design, Russian Academy of Sciences, Odintsovo, Moscow Region, Russia

Abstract. The problem of finding spatial and temporal mappings for affine programs that ensure the locality of data use is considered in context of parallelizing loop nests for execution on universal multi-core processors. A source-to-source translator architecture is proposed for performing parallelizing transformations of affine programs with support for OpenMP technology. Experimental results of studying the performance of parallelized programs from the polybench test suite are presented: lu, syr2k, gramschmidt. The performance of parallel programs obtained using the developed translator ilpy and the modern translator pluto 0.11.4 is compared.

Keywords: *affine programs, polyhedral model, space and time mappings, translator*.
DOI: 10.14357/20790279230405 **EDN: WNRRTT**

References

1. *Banerjee Utpal.* Loop transformations for restructuring compilers: the foundations. Springer Science & Business Media, 2007.
2. *Steinberg B.Ya., Nis Z.Ya., Petrenko V.V., Cherdantsev D.N., Steinberg R.B., and Shulzhenko A.M.* 2006. Sostoyanie I vozmozhnosti otkrytoy rasparallelivayushey sistemy (leto 2006 g.) [State and features of the open parallelizing system (summer 2006)]. Novosibirksk, Akademgorodok. 28-29.
3. *Griebl, Martin.* Automatic parallelization of loop programs for distributed memory architectures. Passau, Germany: Univ. Passau, 2004.
4. *Lengauer, Christian.* “Loop parallelization in the polytope model.” International Conference on Concurrency Theory. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993.
5. *Voevodin V.V., and Voevodin Vl.V.* “Parallel computations.” BHV, Saint-Petersburg (2002).
6. *Lebedev A.S.* 2015. Prostranstvenno-vremennye preobrazovaniya pri rasparallelivanii lineynykh programm [On choosing space-time mappings during automatic parallelization of loop nests with static control flow]. Informatsionnye tekhnologii I vychislitelnye sistemy [Journal of Information Technologies and Computing Systems] 1:19–32.
7. *Lebedev A.S., Solodovnikov V.I.* 2022. Programma dlya postroeniya affinnykh preobrazovaniy program lineynogo klassa [The program to find affine mappings for affine programs]. Patent RF No. 2022667001.
8. *Bastoul, Cédric.* “Openscop: A specification and a library for data exchange in polyhedral compilation

tools.” Paris-Sud University, France, Tech. Rep 9 (2011): 22.
9. *Bastoul, Cédric.* “Extracting polyhedral representation from high level languages.” Tech. rep. Related to the Clan tool. LRI, Paris-Sud University (2008).
10. *Bastoul, Cédric, and Louis-Noël Pouchet.* Candl: the chunky analyzer for dependences in loops. tech. rep., LRI, Paris-Sud University, France, 2012.
11. *Bastoul, Cédric.* “Code generation in the polyhedral model is easier than you think.” Proceedings. 13th International Conference on Parallel Architecture and Compilation Techniques, 2004. PACT 2004. IEEE, 2004.
12. *Loechner, Vincent.* “PolyLib: A library for manipulating parameterized polyhedra.” (1999).
13. *Makhorin A.* GLPK (GNU linear programming kit). Available at: http://www.gnu.org/s/glpk/glpk.html (accessed August 18, 2023).
14. OpenBLAS: an optimized BLAS library. Available at: https://www.openblas.net/ (accessed August 18, 2023).
15. *Yuki T, Pouchet L.* PolyBench 4.2. Available at: https://sourceforge.net/projects/polybench/ (accessed August 18, 2023).
16. *Likhoded N.A.* 2008. Methods for parallelization of loop nests. (In Russian)
17. *Verdoolaege, Sven.* “isl: An integer set library for the polyhedral model.” International Congress on Mathematical Software. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.
18. *Griebl, Martin, Paul Feautrier, and Armin Größlinger.* “Forward communication only placements and their use for parallel program construction.” Languages and Compilers for Parallel Computing: 15th Workshop, LCPC 2002, College Park, MD, USA, July 25-27, 2002. Revised Papers 15. Springer Berlin Heidelberg, 2005.
19. *Bondhugula, Uday, and Albert Cohen.* “Handling Negative Coefficients in Automatic Transformation Schedules.” (2014).
20. *Bondhugula, Uday, 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.

Lebedev A.S. Junior research assistant, Center of Information Technologies in Design, Russian Academy of Sciences, Odintsovo, Moscow Region, 143003, Russian Federation. e-mail: tementy@gmail.com.

Solodovnikov V.I. PhD, director, Center of Information Technologies in Design, Russian Academy of Sciences, Odintsovo, Moscow Region, 143003, Russian Federation. e-mail: info@ditc.ras.ru.