https://doi.org/10.32362/2500-316X-2019-7-5-7-19

УДК 004.4’422

Система автоматического распараллеливания линейных программ для машин с общей и распределенной памятью

Ш.Г. Магомедов@,
А.С. Лебедев

МИРЭА – Российский технологический университет, Москва 119454, Россия
@Автор для переписки, e-mail: msgg@list.ru

Эффективное программирование параллельных архитектур всегда было сложной задачей и особенно усложняется при их современном разнообразии. Задача автоматического распараллеливания программного кода была сформулирована с момента появления первых параллельных отечественных вычислителей (например, ПС2000). К настоящему времени разработаны языки и технологии программирования, которые упрощают работу программиста (Т-Система, MC#, Erlang, Go, OpenCL), но не делают распараллеливание автоматическим. Сложившаяся ситуация требует разработки и развития эффективных инструментов программирования вычислительных систем. Такие инструменты должны поддерживать разработку параллельных программ для систем с общей и распределенной памятью. В работе рассматривается задача автоматического распараллеливания линейных программ для таких систем. Обсуждаются разработанные методы вычисления пространственно-временных преобразований, оптимизирующих локальность программы. Рассматривается реализация методов на языке Haskell в рамках source-to-source транслятора, осуществляющего автоматическое распараллеливание. Осуществляется сравнение быстродействия параллельных программ lu, atax, syr2k, полученных с помощью разработанной системы и современного инструмента Pluto. Эксперименты проводились на двух машинах архитектуры x86_64, объединенных сетью InfiniBand. В качестве технологий распараллеливания использовались OpenMP и MPI. Быстродействие результирующей параллельной программы свидетельствует о практической применимости разработанной системы распараллеливания линейных программ.

Ключевые слова: автоматическое распараллеливание, линейные программы, модель многогранников, оптимизация локальности, линейное целочисленное программирование.

Для цитирования: Магомедов Ш.Г., Лебедев А.С. Система автоматического распараллеливания линейных программ для машин с общей и распределенной памятью // Российский технологический журнал. 2019. Т. 7. № 5. С. 7–19. https://doi.org/10.32362/2500-316X-2019-7-5-7-19

A tool for automatic parallelization of affine programs for systems with shared and distributed memory

Shamil G. Magomedov[@], Artem S. Lebedev

MIREA – Russian Technological University, Moscow 119454, Russia
[@]Corresponding author, e-mail: msgg@list.ru

Effective programming of parallel architectures has always been a challenge, and it is especially complicated with their modern diversity. The task of automatic parallelization of program code was formulated from the moment of the appearance of the first parallel computers made in Russia (for example, PS2000). To date, programming languages and technologies have been developed that simplify the work of a programmer (T-System, MC#, Erlang, Go, OpenCL), but do not make parallelization automatic. The current situation requires the development of effective programming tools for parallel computing systems. Such tools should support the development of parallel programs for systems with shared and distributed memory. The paper deals with the problem of automatic parallelization of affine programs for such systems. Methods for calculating space-time mappings that optimize the locality of the program are discussed. The implementation of developed methods is done in Haskell within the source-to-source translator performing automatic parallelization. A comparison of the performance of parallel variants of lu, atax, syr2k programs obtained using the developed tool and the modern Pluto tool is made. The experiments were performed on two x86_64 machines connected by the InfiniBand network. OpenMP and MPI were used as parallelization technologies. The performance of the resulting parallel program indicates the practical applicability of the developed tool for affine programs parallelization.

Keywords: automatic parallelization, affine programs, polyhedral model, locality optimization, integer linear programming.

For citation: Magomedov Sh.G., Lebedev A.S. A tool for automatic parallelization of affine programs for systems with shared and distributed memory. *Rossiiskii tekhnologicheskii zhurnal* = Russian Technological Journal. 2019;7(5):7-19 (in Russ.). https://doi.org/10.32362/2500-316X-2019-7-5-7-19

Введение

Вычислительно емкие научные и инженерные приложения в большинстве своем тратят значительную часть процессорного времени на исполнение вложенностей циклов, часто удовлетворяющих свойствам линейных программ. Для анализа и преобразования таких программ существует модель многогранников [1], позволяющая применить техники распараллеливания и локализации данных с целью повышения быстродействия критических вычислительно емких участков.

Среди первых проектов, в которых были реализованы методы модели многогранников, выделяют LooPo [2] и PIPS [3]. Дальнейшее развитие методы модели многогранников нашли в современном source-to-source трансляторе Pluto [4, 5] и работах, фокусирующихся на агрегировании итераций в блоки для дополнительного улучшения временной и пространственной локальности вычислений [6, 7]. В работе [8] авторы применили методы машинного обучения для выбора оптимизирующих преобразований в модели многогранников, сделав шаг в сторону от распространенной практики применения

классических методов оптимизации. Практическая ценность методов модели многогранников для систем трансляции предметно-ориентированных языков (DSL) была показана в проекте Pencil [9], при этом остается актуальным традиционный вариант использования распараллеливающих систем: код, написанный на языке C или FORTRAN, оптимизируется для параллельного выполнения и компилируется для целевой архитектуры, о чем свидетельствуют направления развития open-source [4, 10, 11] и коммерческих компиляторов [12, 13].

Распараллеливание программ в модели многогранников осуществляется в пять этапов:

1. Анализируются зависимости по данным: потоковые зависимости (чтение после записи), антизависимости (запись после чтения), зависимости через выход (запись после записи).
2. Вычисляется многомерное расписание программы, обладающее параллелизмом и оптимизирующее временную локальность данных при вычислениях.
3. Вычисляется пространственное размещение, обладающее параллелизмом и оптимизирующее пространственную локальность данных при вычислениях.
4. Выполняется агрегирование итераций в блоки для дополнительного улучшения временной и пространственной локальности вычислений.
5. Генерируется параллельная программа согласно подходу [14] в соответствии с пространственно-временными преобразованиями, вычисленными на этапах 2–4.

Целью настоящей работы является практическая реализация ранее разработанных методов вычисления пространственно-временных преобразований [15–17] в рамках source-to-source транслятора линейных программ, написанных на языке C. Аналогом для сравнения служит инструмент Pluto – развитый source-to-source транслятор, реализующий методы [4, 5]. Вначале приводится понятийный аппарат модели многогранников, затем особенности реализации разработанной системы, после чего проводится сравнение системы с инструментом Pluto на примере трех алгоритмов линейной алгебры: LU-разложение квадратной матрицы, матричные произведения atax и syr2k.

Понятийный аппарат модели многогранников

Модель многогранников – модель последовательных и параллельных вычислений, разработанная для анализа и преобразования линейных программ. Поиск и применение оптимизирующих преобразований осуществляется в рамках математической модели («полиэдрального представления») без непосредственных манипуляций с исходным текстовым представлением программ. Класс линейных программ описывается следующими ограничениями:

- в качестве исполнительных операторов используются только операторы присваивания, правая часть которых – арифметическое выражение;
- для описания повторяющихся операций используются только циклы с известным числом повторений (циклы DO языка FORTRAN или эквивалентные циклы for языка C); допускаются вложенности циклов произвольной структуры (плотные и неплотные); границы изменения счетчиков циклов задаются аффинными функциями от индексов циклов, обрамляющих данный цикл, и внешних переменных программы;
- допускается использование операторов ветвления, в условиях которых используются только аффинные функции от индексов циклов, обрамляющих оператор, и внешних переменных программы; не допускаются побочные выходы из циклов;

• допустимой структурой данных является многомерный массив; индексы массива являются аффинными функциями от индексов циклов, обрамляющих оператор, и внешних переменных программы;

• внешние переменные программы являются целочисленными, и их конкретные значения известны только во время работы программы.

Пусть Ω — множество операций над памятью, которые требуется выполнить; $\theta(u)$ $u \in \Omega$ – функция расписания вычислений, задающая логическое время выполнения операций. Тогда множество операций $F(t) = \{u \in \Omega \mid \theta(u) = t\}$ называется фронтом расписания на t. Пусть $L = \max\limits_{u \in \Omega}(\theta(u))$ – задержка расписания. Тогда программа с синхронным параллелизмом может быть сконструирована следующим образом.

Для систем с общей памятью:

1 ДЛЯ ($t = 0; t \le L; t = t + 1$)
2 ВЫПОЛНИТЬ_ПАРАЛЛЕЛЬНО $F(t)$
3 БАРЬЕР_СИНХРОНИЗАЦИИ
4 КОНЕЦ

Для систем с распределенной памятью:

1 ДЛЯ ($t = 0; t \le L; t = t + 1$)
2 ИНФОРМАЦИОННЫЙ_ОБМЕН($\forall r_{\text{read}} \in \Omega$, где r_{read} – операция удаленного чтения
3 ВЫПОЛНИТЬ_ПАРАЛЛЕЛЬНО $F(t)$
4 БАРЬЕР_СИНХРОНИЗАЦИИ
5 ИНФОРМАЦИОННЫЙ_ОБМЕН($\forall r_{\text{write}} \in \Omega$, где r_{write} – операция удаленной записи)
6 КОНЕЦ

Каждой инструкции X соответствует многогранник D_X на множестве $\mathbb{Q}^{p_X}$, называемый доменом, где p_X – размерность ее итерационного пространства (количество циклов в программе, включающих X). Операции $u \in \Omega$, определяемые инструкцией X, представляются 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 длиной q_z. В модели многогранников вводятся следующие аффинные преобразования для операций: $\theta(\langle \vec{i}, \vec{z}; X \rangle) = \theta_X(\vec{i}, \vec{z}) = \vec{v}_X \cdot \vec{i} + \vec{v}'_X \cdot \vec{z} + v^0_X$, $v^0_X \in \mathbb{Z}$, $\vec{v}'_X \in \mathbb{Z}^{q_z}$, $\vec{v}'_X \in \mathbb{Z}^{q_z}$, $\vec{i} \in D_X$ – функции расписания вычислений, $\pi(\langle \vec{i}, \vec{z}; X \rangle) = \pi_X(\vec{i}, \vec{z}) = \vec{v}_X \cdot \vec{i} + \vec{v}'_X \cdot \vec{z} + v^0_X$, $v^0_X \in \mathbb{Z}$, $\vec{v}'_X \in \mathbb{Z}^{q_z}$, $\vec{v}'_X \in \mathbb{Z}^{q_z}$, $\vec{i} \in D_X$ – функции размещения вычислений. Каждому массиву A соответствует множество допустимых индексов $D_A \subset \mathbb{Z}^{p_A}$, где p_A – размерность массива A. Вводятся также аффинные преобразования для элементов данных: $\eta(\langle \vec{g}, \vec{z}; X \rangle) = \eta_A(\vec{g}, \vec{z}) = \vec{v}_A \cdot \vec{g} + \vec{v}'_A \cdot \vec{z} + v^0_A$, $v^0_A \in \mathbb{Z}$, $\vec{v}_A \in \mathbb{Z}^{p_A}$, $\vec{v}'_A \in \mathbb{Z}^{q_z}$, $\vec{g} \in D_A$ – функции размещения данных. Все аффинные преобразования являются функциями с областью значений $\mathbb{N}_0$. Модель многогранников оперирует понятием пространства виртуальных процессоров, количество которых не ограничено. Для сопоставления виртуальным процессорам операций, которые они должны выполнять, и элементов данных, которыми они должны владеть, и служат функции размещения вычислений и данных соответственно. Функции расписания и размещения вычислений называются пространственно-временными преобразованиями.

Если между двумя операциями $\sigma \in \Omega$ и $\delta \in \Omega$ существует зависимость по данным, и δ не может быть выполнена раньше σ, то функция расписания вычислений $\theta(u)$, $u \in \Omega$ должна удовлетворять принципу причинности: $\theta(\delta) - \theta(\sigma) \ge 1$. Уменьшение разницы в левой части неравенства (сокращение задержки использования данных) ведет к улучшению временной локальности вычислений.

Размещение вычислений и данных обладает свойством вперед направленных коммуникаций тогда и только тогда, когда все направления коммуникаций исходят от виртуальных процессоров с меньшим индексом к виртуальным процессорам с большим индексом. Применение размещений, обладающих таким свойством, представляет практический интерес: в работе [18] приводятся случаи, в которых размещение с таким свойством позволяет вдвое уменьшить количество партнеров коммуникации. Для систем с общей памятью это свойство обеспечивается условием $\pi(\delta)-\pi(\sigma)\geq 0$, для систем с распределенной памятью – двумя: $\pi(r_{\text{read}})-\eta(\langle \vec{g}(r_{\text{read}}),\vec{z};A_{r_{\text{read}}}\rangle)\geq 0$ для операции удаленного чтения $r_{\text{read}}\in\Omega$ и $\eta(\langle \vec{g}(r_{\text{write}}),\vec{z};A_{r_{\text{write}}}\rangle)-\pi(r_{\text{write}})\geq 0$ для операции удаленной записи $r_{\text{write}}\in\Omega$. Уменьшение разницы в левой части каждого неравенства (сокращение расстояния использования данных) ведет к улучшению пространственной локальности вычислений.

В работах [15, 16] верхние пороги задержки и расстояния использования данных для каждой информационной зависимости используются как критерии качества оптимизации локальности вычислений. Построение пространственно-временных преобразований сводится к задачам линейного целочисленного программирования. Предлагается линейная свертка критериев в качестве целевой функции при решении задачи оптимизации: весовые коэффициенты имеют большие значения для тех информационных зависимостей, которые чаще возникают.

Информационный обмен между параллельными процессами

Вычислительное оборудование имеет конечное число процессорных элементов, поэтому возникает задача сопоставления виртуальных процессоров физическим. Пусть вычислительная система имеет Q физических процессоров с индексами $r=0,1,\ldots,Q-1$. Пусть v ($v\in\mathbb{N}_0$) – индекс виртуального процессора, $r(v)$ – индекс физического процессора, которому виртуальный процессор с индексом v поставлен в соответствие. В работах [19–21] упоминались два применяемых на практике способа распределения виртуальных процессоров между физическими: блочное, если $r(v)=\lfloor v/Q\rfloor$, и циклическое, если $r(v)=v-Q\lfloor v/Q\rfloor$.

Используемый в работе метод организации информационного обмена между параллельными процессами опирается на блочную схему распределения. Физический процессор с индексом r обрабатывает фрагмент пространства виртуальных процессоров $l(r),\ldots,u(r)$:

$$l(r+1)=u(r)+1,$$

$$L\leq l(r)\leq u(r)\leq U,\quad l(r)=L+r\left\lfloor\frac{U-L+Q}{Q}\right\rfloor,$$

$$L=\min\left(\min_X(\pi_X),\min_A(\eta_A)\right),\quad U=\max\left(\max_X(\pi_X),\max_X(\eta_A)\right).$$

Операции удаленного чтения и удаленной записи индуцируют информационный обмен между физическими процессорами. Пусть процессор R осуществляет доступ к данным, расположенным на процессоре r, и найденные функции размещения вычислений и данных удовлетворяют свойству вперед направленных коммуникаций [20]. Тогда выполняется $R > r$ для операций удаленного чтения и $R < r$ для операций удаленной записи. Информационный обмен для операций удаленного чтения выполняется перед выполнением фронта операций, а для операций удаленной записи – после выполнения фронта опе-

раций. Двухсторонняя коммуникация процессов требует библиотеки MPI и применяет только две функции – MPI_Send и MPI_Recv. Шаги 2 и 5 схемы параллельной программы для систем с распределенной памятью реализуются следующим образом:

```
// 2 ИНФОРМАЦИОННЫЙ_ОБМЕН
ДЛЯ ( j = 1; j <= n; j = j + 1 )
 ЕСЛИ ( a_j ∈ {r_read} ) ТО
  ДЛЯ ( r = 0; r < R; r = r + 1 )
   MPI_Recv(данные от r к R)
  КОНЕЦ
  ДЛЯ ( r = R + 1; r < Q; r = r + 1 )
   MPI_Send(данные от R к r)
  КОНЕЦ
 КОНЕЦ
КОНЕЦ
```

```
// 5 ИНФОРМАЦИОННЫЙ_ОБМЕН
ДЛЯ ( j = 1; j <= n; j = j + 1 )
 ЕСЛИ ( a_j ∈ {r_write} ) ТО
  ДЛЯ ( r = 0; r < R; r = r + 1 )
   MPI_Recv(данные от r к R)
  КОНЕЦ
  ДЛЯ ( r = R + 1; r < Q; r = r + 1 )
   MPI_Send(данные от R к r)
  КОНЕЦ
 КОНЕЦ
КОНЕЦ
```

Здесь R – собственный ранг MPI-процесса, r используется для индексирования рангов других MPI-процессов.

Архитектура системы автоматического распараллеливания

На рисунке представлена диаграмма потоков данных процесса распараллеливания. Входными данными является программа на языке C, содержащая вложенности циклов, которые требуется оптимизировать для параллельного выполнения. С помощью инструмента clan [22] входная программа преобразуется в полиэдральное представление OpenSCOP [23]. Анализ информационных зависимостей осуществляется инструментом candl [24]. Полиэдральное представление OpenSCOP, содержащее информацию об инструкциях, доменах, доступах к массивам, информационных зависимостях, подается на вход программе ilp_parallelizer, написанной на языке Haskell, которая реализует ранее разработанные методы вычисления пространственно-временных преобразований [15, 16] и организации информационного обмена между MPI-процессами [17].

Программа ilp_parallelizer извлекает из представления OpenSCOP информацию об обобщенном графе зависимостей, которая исчерпывающе описывает поток данных в программе. Эта информация сохраняется во внутренней структуре, названной графом потока данных (DFG). В дальнейшем структура DFG используется в качестве промежуточного представления программы, а представление OpenSCOP используется только для взаимодействия с внешней программой cloog [14].

После формирования структуры DFG вычисляется многомерное расписание программы, которое в дальнейшем будет использоваться при формировании OpenSCOP для финальной кодогенерации (в случае систем с общей памятью) или для дальнейших преобразований программы (в случае систем с распределенной памятью). Размещение вычислений должно быть линейно независимым от компонентов многомерного расписания [15], поэтому вычисляется после того, как будет вычислено многомерное расписание.

Для систем с распределенной памятью размещение вычислений и данных происходит совместно, по методу [16]. Вычисленное размещение данных совместно с многомерным расписанием используются в качестве распределяющих функций (scattering functions) при формировании представления OpenSCOP и дальнейшей кодогенерации (применения

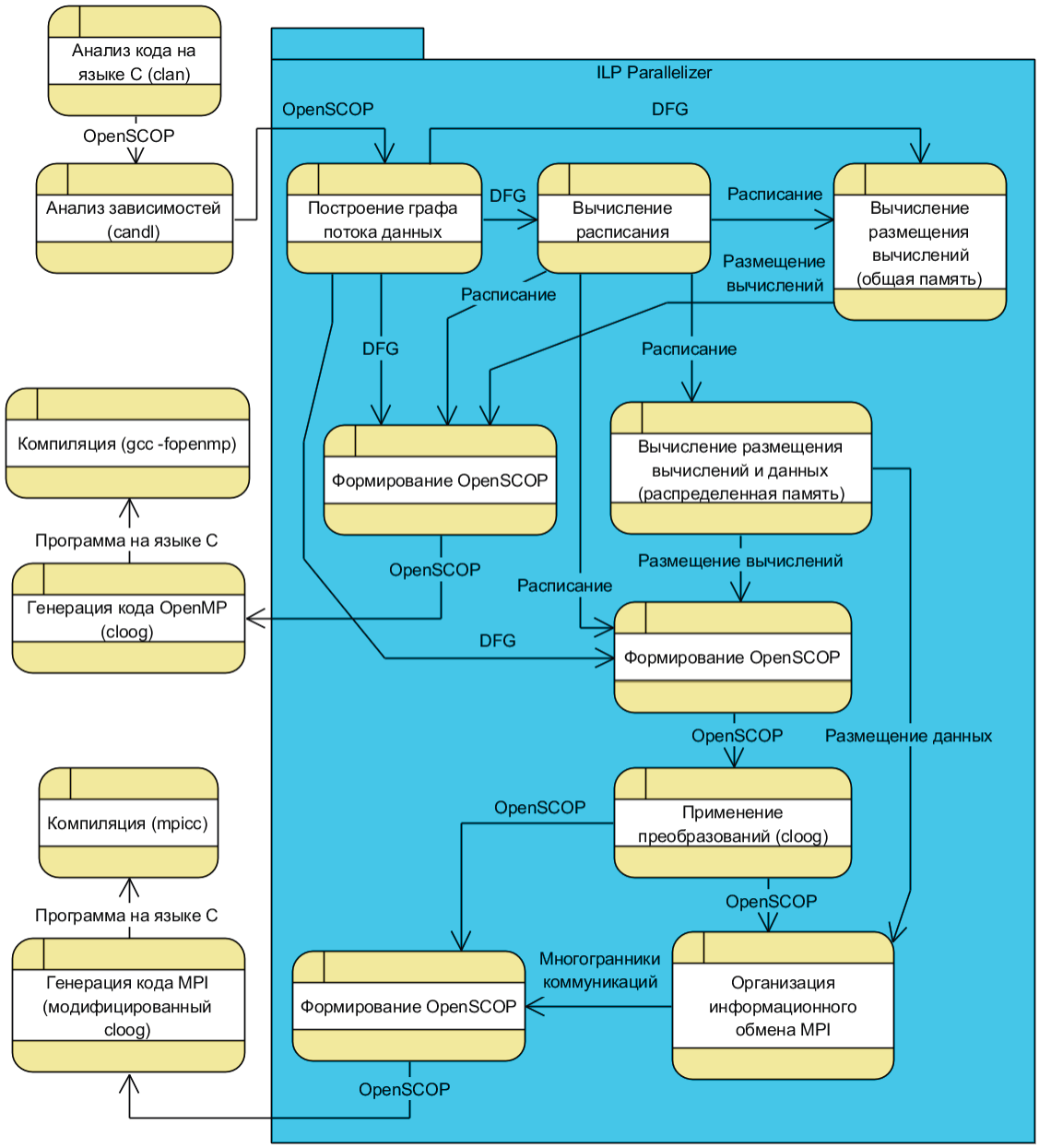


Диаграмма потоков данных процесса распараллеливания.

пространственно-временных преобразований) с помощью программы cloog. Полученная параллельная программа также анализируется, и вычисляются многогранники коммуникаций по методу [17] на основе ранее вычисленного размещения данных. Для генерации MPI-кода cloog был модифицирован так, чтобы извлекать из представления OpenSCOP информацию о многогранниках коммуникаций и на ее основе инструментировать программу MPI-вызовами.

С помощью функций MPI_Send и MPI_Recv можно передавать и принимать данные, занимающие непрерывную область памяти. В общем случае многогранники коммуникаций могут описывать данные, расположенные в памяти разрозненно. Поэтому в общем случае применимы техники упаковки информационных пакетов в форматах хранения разреженных матриц. Для распространенного на практике случая, когда двухмерный многогранник коммуникаций представлен совокупностью одномерных полос различной

длины (совокупность непрерывных фрагментов строк матрицы), применение упаковки в форматы разреженных матриц избыточно. Разработанная система генерирует код, в котором каждый непрерывный одномерный фрагмент данных обрабатывается отдельной парой вызовов MPI_Send и MPI_Recv. При этом сообщение помечается тегом $T_{A_k} + i$, где T_{A_k} – уникальная для каждого двумерного массива A_k целочисленная константа, имеющая значение $k \cdot 10^s$, а i – индекс строки $A_k[i]$, в которой расположен фрагмент для приема и передачи. Значение s выбирается так, чтобы $10^s > len(A_k)$ для всех матриц A_k, участвующих в информационном обмене. В целях упрощения экспериментальной реализации системы в каждом MPI-процессе память выделяется для хранения массивов целиком.

Особенности практической реализации методов вычисления пространственно-временных преобразований

Проблема возрастающей сложности задачи линейного целочисленного программирования при увеличении количества переменных и ограничений не позволяет вычислять неотрицательные пространственно-временные преобразования распараллеливаемых программ за приемлемое время. В работах [15, 16] подробно описывается процедура построения системы ограничений с применением леммы Фаркаша для выписывания неотрицательного шаблона функции расписания или размещения вычислений. Для каждого измерения многогранника D_X есть минимум две ограничивающих гиперплоскости (верхняя и нижняя границы изменения счетчика соответствующего цикла). В этом случае фигурируют не менее $2p_X$ множителей Фаркаша и добавляется столько же неравенств для ограничения их неотрицательности. Операторы ветвления могут индуцировать дополнительные гиперплоскости и ограничения соответственно. Уменьшить количество переменных и ограничений можно, отказавшись от свойства неотрицательности пространственно-временных преобразований, оставив общий вид для всех аффинных отображений $\phi_X\left(\vec{i}_X, \vec{z}\right) = \vec{v}_X \cdot \vec{i}_X + \vec{v}'_X \cdot \vec{z} + v^0_X, v^0_X \in \mathbb{Z}, \vec{v}_X \in \mathbb{Z}^{p_X}, \vec{v}'_X \in \mathbb{Z}^{q_z}$. В этом случае фигурируют $p_X + q_z + 1$ переменных и не добавляется неравенств. Несмотря на то, что выигрыш в количестве переменных достигается при условии $q_z + 1 < p_X + \delta$, где δ – количество дополнительных гиперплоскостей, индуцированных операторами ветвления, выигрыш в количестве неравенств безусловен, это позволяет снизить сложность задачи линейного целочисленного программирования и получать пространственно-временные преобразования за время, меньшее секунды, на процессоре уровня Intel Core i5 3337U. Применение этой техники дает корректные преобразования, но их использование при кодогенерации дает существенные издержки пересчета индексов циклов из-за возможных отрицательных значений, что ведет к громоздким арифметическим выражениям, увеличивающим вычислительную сложность распараллеленной программы, и затрудняет визуальный анализ результата распараллеливания при отладке. Для решения этой проблемы принято решение нормировать полученные пространственно-временные преобразования.

Величина $\Theta_{\text{offset}} = -\min_X\left(\min_{D_X} \theta_X\right)$ добавляется ко всем функциям расписания вычислений. Величина $\Pi_{\text{offset}} = -\min\left(\min_X\left(\min_{D_X} \pi_X\right), \min_A\left(\min_{D_A} \eta_A\right)\right)$ добавляется ко всем функциям размещения вычислений и данных. Минимумы вычисляются с применением аппарата линейного целочисленного программирования.

Примеры распараллеливания программ

Разработанные методы автоматического распараллеливания линейных программ были применены к трем реализациям алгоритмов линейной алгебры: LU-разложение квадратной матрицы, матричные произведения atax и syr2k.

Каждая программа была распараллелена для систем с общей памятью с помощью OpenMP, а для систем с распределенной памятью – с помощью MPI. Вычисления производились с двойной точностью (тип double языка C). Эксперименты проводились на двух машинах кластера Института комплексной безопасности и специального приборостроения МИРЭА – Российского технологического университета (табл. 1). Запуск OpenMP-программ осуществлялся на машине 1, запуск MPI-программ – на обеих машинах. Сравнивалась производительность параллельной программы, полученной с применением разработанной системы и современного компилятора Pluto 0.11.4 [4] (для MPI использовалась ветка distmem [5]). Ускорение оценивалось относительно запуска последовательного варианта программы на машине 1.

Разработанная система и современный компилятор pluto (ветка distmem) генерируют OpenMP-код в MPI-программах с целью снизить расходы оперативной памяти и уменьшить объем коммуникации между MPI-процессами. Однако установленная на вычислительном оборудовании реализация OpenMPI, поставляемая с драйверами InfiniBand-адаптеров Mellanox, не позволила получить выигрыш в производительности за счет множественных нитей OpenMP внутри MPI-процесса. Напротив, наблюдалось неприемлемое на практике снижение производительности, поэтому было отдано предпочтение отдельным MPI-процессам вместо нитей OpenMP.

В табл. 2 приведены результаты распараллеливания трех линейных программ. Варианты параллельных программ, полученные с применением разработанной системы, помечены как ILP, а варианты, полученные с применением компилятора Pluto, помечены

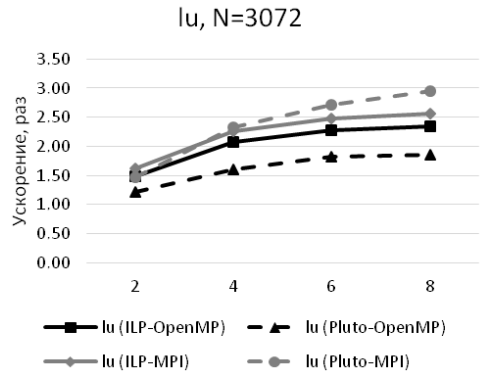
Таблица 1. Конфигурация оборудования и ключи компиляции программ

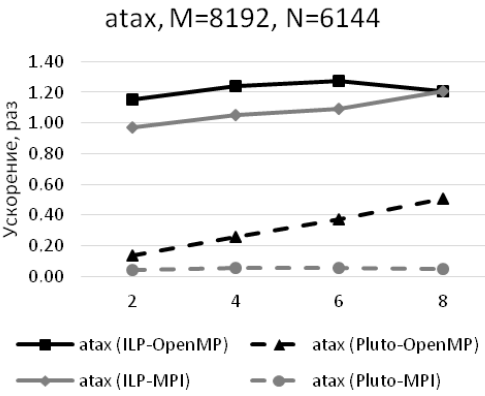
Конфигурация машины 1 (node0)	Intel(R) Xeon(R) CPU E5-2690 v2 @ 3.00GHz, 8GB RAM, InfiniBand: Mellanox Technologies MT26428 [ConnectX VPI PCIe 2.0 5GT/s - IB QDR / 10GigE] (rev b0)
Конфигурация машины 2 (node1)	Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz, 4GB RAM, InfiniBand: Mellanox Technologies MT27500 Family [ConnectX-3]
Сеть передачи сообщений	InfiniBand 40Gb/sec
Операционная система	Linux CentOS 7.3 x64
Компилятор	gcc 4.8.5
Реализация MPI	openmpi-1.10.5a1
Компиляция OpenMP-программ	gcc -O2 -std=c99 -fopenmp
Компиляция MPI-программ (ILP)	mpicc -O2 -std=c99 -fopenmp
Компиляция MPI-программ (pluto)	Для программы lu: mpicc -O2 -fopenmp -D__MPI -DTIME \ -D__DONT_USE_INV_BLOCK_DIST_FUNCTION Для программ atax и syr2k: mpicc -O2 -fopenmp -D__MPI -DTIME
Трансляция pluto для OpenMP	polycc --parallel
Трансляция pluto-distmem для MPI	polycc --distmem --mpiomp --commopt_fop --isldep --lastwriter --cloogsh --timereport
Схема распределения процессов при запуске MPI-программ	2 процесса: mpirun -np 2 -host node0,node1 4 процесса: mpirun -np 4 -host \node0,node0,node1,node1 6 процессов: mpirun -np 6 -host \node0,node0,node0,node1,node1,node1 8 процессов: mpirun -np 8 -host \node0,node0,node0,node0,node1,node1,node1,node1

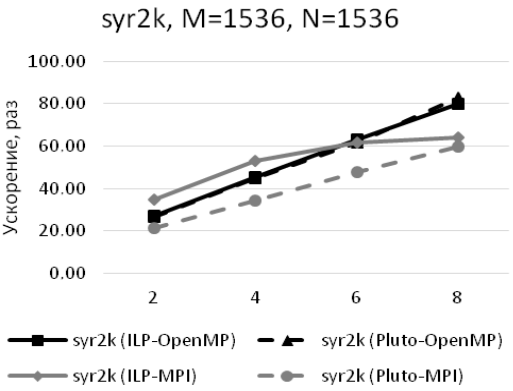
как Pluto. Параллельный код запускался для исполнения в 2, 4, 6, 8 потоках. В экспериментах с OpenMP работали множественные нити внутри одного процесса, запущенного на машине 1.

Таблица 2. Результаты распараллеливания программ

Последовательная программа	Результаты распараллеливания

// lu
for (int k = 0; k < N; k++) {
for (int l = k + 1; l < N; l++)
A[l][k] /= A[k][k];
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];
}
	

// atax
for (int i = 0; i < N; i++)
y[i] = 0;
for (int i = 0; i < M; i++) {
tmp[i] = 0;
for (int j = 0; j < N; j++)
tmp[i] = tmp[i] + A[i][j] * x[j];
for (int j = 0; j < N; j++)
y[j] = y[j] + A[i][j] * tmp[i];
}
	

// syr2k
for (int i = 0; i < N; i++) {
for (int j = 0; j <= i; j++)
C[i][j] *= beta;
for (int k = 0; k < M; k++)
for (int j = 0; j <= i; j++)
C[i][j] += alpha * (A[j][k] * B[i][k] + B[j][k] * A[i][k]);
}
	

В экспериментах с MPI работали единственные нити внутри множественных MPI-процессов, запущенных на обеих машинах (количество процессов разделялось поровну между машинами). Рассматриваемый в качестве аналога разработанной системе компилятор Pluto на момент написания статьи находится в стадии экспериментального прототипа, и потому его работа нестабильна. Результирующий параллельный код программы lu удалось успешно скомпилировать только с дополнительным определением define (табл. 1). Присутствие этого определения для других рассматриваемых примеров

atax и syr2k приводило к заметному снижению быстродействия и потому не применялось при сравнении быстродействия параллельных вариантов программ.

Несмотря на присутствие нежелательных параметров сборки кода, вариант параллельной программы lu (Pluto-MPI) оказался на 14% быстрее варианта lu (ILP-MPI), показав ускорение в 2.94 раза против 2.56 раза при выполнении в 8 потоков. Выигрыш был достигнут за счет более равномерной загрузки процессов работой, осуществляемой динамически, в отличие от статического подхода, применяемого в разработанной системе, при котором допускается простой процессов. Вариант lu (ILP-OpenMP) оказался на 27% быстрее варианта lu (Pluto-OpenMP), показав ускорение в 2.35 раза против 1.85 раза при выполнении в 8 потоков за счет применения разработанного метода оптимизации пространственной и временной локальности данных.

Для программы atax все результаты, полученные с применением компилятора pluto, показали неприемлемое на практике ухудшение быстродействия. Вариант atax (ILP-OpenMP) показал наилучшее быстродействие при выполнении в 6 потоков – ускорение в 1.27 раза. Вариант atax (ILP-MPI) быстрее отработал в 8 потоков – ускорение в 1.2 раза. Примечательно, что оба варианта показали сходную производительность при выполнении в 8 потоков.

Для программы syr2k совпали результаты syr2k (ILP-OpenMP) и syr2k (Pluto-OpenMP), поскольку системы сгенерировали одинаковый код. Расхождения в графиках обусловлены влиянием других процессов операционной системы на вычисления. Для выполнения в 8 потоков было показано ускорение в 80 раз за счет оптимизации локальности программы. Вариант syr2k (ILP-MPI) отказался быстрее варианта syr2k (Pluto-MPI) во всех запусках. Наибольшая разница в производительности – 53% – была достигнута при выполнении в 4 потока (ускорение в 53.03 раза против 34.51 раза). При увеличении количества потоков разрыв сокращался и составил 7% (ускорение в 63.97 раза против 59.7 раза) при исполнении в 8 потоков. При увеличении количества нитей вариант syr2k (Pluto-MPI) сохраняет масштабируемость за счет динамической загрузки процессов работой, а при уменьшении количества нитей вариант syr2k (ILP-MPI) выигрывает за счет лучшей оптимизации локальности данных.

Заключение

Разработанная система автоматического распараллеливания линейных программ на примере алгоритмов lu, atax, syr2k показала практическую применимость. Производительность полученной параллельной программы в большинстве случаев превзошла результаты современного распараллеливающего транслятора Pluto. Выигрыш в производительности обусловлен применением разработанных методов вычисления пространственно-временных преобразований, основанных на более гибком способе определения качества альтернатив (линейная свертка критериев), чем у Pluto (лексикографическое упорядочивание). Эксперименты показали необходимость совершенствования механизмов загрузки процессов работой с целью дальнейшего улучшения быстродействия MPI-программ. Техники распределения вычислений и данных по машинам остаются предметом дальнейших исследований.

Благодарности

Данные исследования проведены при поддержке РТУ МИРЭА в рамках инициативной научно-исследовательской работы ИЦМР-5 «Разработка автоматизированной системы управления закупками».

Список литературы / References:

1. Griebl M., Lengauer C. On the space-time mapping of WHILE-loops. *Parallel Processing Lett.* 1994; 4(3):221-232. https://doi.org/10.1142/S0129626494000223

2. Griebl M., Lengauer C. The loop parallelizer LooPo – announcement. *Int. Workshop on Languages and Compilers for Parallel Computing.* Springer, Berlin, Heidelberg, 1996. P. 603-604. https://doi.org/10.1007/BFb0017283

3. Irigoin F., Jouvelot P., Triolet R. Semantical interprocedural parallelization: An overview of the PIPS project. Proceed. of the 5th Int. Conf. on Supercomputing, ACM, New York, ICS '91. 1991. P. 244-251. https://doi.org/10.1145/109025.109086

4. Bondhugula U., Hartono A., Ramanujam J., Sadayappan P. A practical automatic polyhedral parallelizer and locality optimizer. *ACM SIGPLAN Notices*. 2008;43(6):101-113. https://doi.org/10.1145/1379022.1375595

5. Bondhugula U. Compiling affine loop nests for distributed-memory parallel architectures. *SC'13: Proceed. of the Int. Conf. on High Performance Computing, Networking, Storage and Analysis.* IEEE, 2013. P. 1-12. https://doi.org/10.1145/2503210.2503289

6. Bondhugula U., Bandishti V., Pananilath I. Diamond tiling: Tiling techniques to maximize parallelism for stencil computations. *IEEE Trans. on Parallel and Distributed Systems.* 2016;28(5):1285-1298. https://doi.org/10.1109/TPDS.2016.2615094

7. Malas T.M., Hager G., Ltaief H., Keyes D. Multidimensional intratile parallelization for memory-starved stencil computations. *ACM Trans. on Parallel Computing (TOPC).* 2018;4(3):12. https://doi.org/10.1145/3155290

8. Park E., Cavazos J., Pouchet L.-N., Bastoul C., Cohen A., Sadayappan P. Predictive modeling in a polyhedral optimization space. *Int. J. Parallel Program.* 2013;41(5):704-750. https://doi.org/10.1007/s10766-013-0241-1

9. Baghdadi R., Beaugnon U., Cohen A., Grosser T., Kruse M., Reddy C., Verdoolaege S., Betts A., Donaldson A.F., Ketema J., Absar J., Van Haastregt S., Kravets A., Lokhmotov A., David R., Hajiyev E. Pencil: A platform-neutral compute intermediate language for accelerator programming. *2015 Int. Conf. on Parallel Architecture and Compilation (PACT).* IEEE, 2015. P. 138-149. https://doi.org/10.1109/pact.2015.17

10. Lee S., Vetter J. S. OpenARC: Extensible OpenACC compiler framework for directive-based accelerator programming study. *Proceed. of the First Workshop on Accelerator Programming using Directives.* IEEE Press, 2014. P. 1-11. http://dx.doi.org/10.1109/WACCPD.2014.7

11. Grosser T., Zheng H., Aloor R., Simbürger A., Größlinger A., Pouchet L.-N. Polly-polyhedral optimization in LLVM. In: Alias C., Bastoul C. (eds.) *Proceed. of the First Int. Workshop on Polyhedral Compilation Techniques (IMPACT)*. INRIA Grenoble Rhône-Alpes, 2011. P. 1.

12. Intel® C++ Compiler 19.0 Developer Guide and Reference. Submitted March 7, 2019. URL: https://software.intel.com/en-us/cpp-compiler-developer-guide-and-reference-enabling-auto-parallelization

13. PGI 2019 Version Information and New Features. URL: https://www.pgroup.com/support/release-2019.htm

14. Bastoul C. Code generation in the polyhedral model is easier than you think. *Proceed. of the 13th Int. Conf. on Parallel Architectures and Compilation Techniques.* IEEE Computer Society, 2004. P. 7-16. https://doi.org/10.1109/PACT.2004.1342537

15. Лебедев А.С. Пространственно-временные преобразования при распараллеливании линейных программ. *Информационные технологии и вычислительные системы.* 2015;(1):19-32.

[Lebedev A.S. Space-time mappings for parallelization of affine programs. *Informatsionnyye tekhnologii i vychislitel'nyye sistemy* [Information Technology and Computing Systems]. 2015;1:19-32 (in Russ.).]

16. Лебедев А.С. Размещение данных при автоматическом распараллеливании линейных программ для систем с распределенной памятью. *Вестник Рыбинского государственного авиационного технического университета имени П.А. Соловьева.* 2015;(3):92-99.

[Lebedev A.S. Construction of data placements for automatic parallelization of affine programs for distributed memory systems. *Vestnik Rybinskogo gosudarstvennogo aviatsionnogo tekhnologicheskogo universiteta im. P.A. Solov'yova* [Bulletin of the P.A. Soloviev Rybinsk State Aviation Technical University]. 2015;(3):92-99 (in Russ.).]

17. Лебедев А.С. Организация информационного обмена между параллельными процессами при автоматическом распараллеливании линейных программ для кластерных систем с применением модели многогранников. *Программные системы: теория и приложения.* 2017;(4):3-20. https://doi.org/10.25209/2079-3316-2017-8-4-3-20

[Lebedev A.S. Organizing communication of parallel processes during automatic parallelization of loop nests with static control flow for cluster systems using polyhedral model. *Programmnyye sistemy: teoriya i prilozheniya* [Software Systems: Theory and Applications]. 2017;(4):3-20 (in Russ.). https://doi.org/10.25209/2079-3316-2017-8-4-3-20]

18. Griebl M., Feautrier P., Größlinger A. Forward communication only placements and their use for parallel program construction. *Int. Workshop on Languages and Compilers for Parallel Computing.* Springer, Berlin, Heidelberg, 2002. P. 16-30. https://doi.org/10.1007/11596110_2

19. Feautrier P. Toward automatic distribution. *Parallel Processing Lett.* 1994;4(3):233-244. https://doi.org/10.1142/S0129626494000235

20. Griebl M. Automatic parallelization of loop programs for distributed memory architectures. Univ. Passau, 2004. 207 p. URL: http://www.infosun.fim.uni-passau.de/cl/publications/docs/Gri04.pdf

21. Reddy C., Bondhugula U. Effective automatic computation placement and data allocation for parallelization of regular programs. *Proceed. of the 28th ACM Int. Conf. on Supercomputing.* ACM, 2014. P. 13-22. https://doi.org/10.1145/2597652.2597673

22. Bastoul C., Cohen A., Girbal S., Sharma S., Temam O. Putting polyhedral loop transformations to work. *Int. Workshop on Languages and Compilers for Parallel Computing.* Springer, Berlin, Heidelberg, 2003. P. 209-225. https://doi.org/10.1007/978-3-540-24644-2_14

23. Bastoul C. Openscop: A specification and a library for data exchange in polyhedral compilation tools. Tech. Rep. Paris-Sud University, France, 2011. V. 9. URL: http://icps.u-strasbg.fr/~bastoul/development/openscop/docs/openscop.pdf

24. Bastoul C., Pouchet L.N. Candl: The chunky analyzer for dependences in loops. Tech. Rep. LRI, Paris-Sud University, France, 2012. URL: http://icps.u-strasbg.fr/~bastoul/development/candl/#DOC

Сведения об авторах:

Магомедов Шамиль Гасангусейнович, кандидат технических наук, доцент кафедры КБ-4 «Автоматизированные системы управления» Института комплексной безопасности и специального приборостроения ФГБОУ ВО «МИРЭА – Российский технологический университет» (119454, Россия, Москва, пр-т Вернадского, д. 78).

Лебедев Артем Сергеевич, преподаватель кафедры КБ-4 «Автоматизированные системы управления» Института комплексной безопасности и специального приборостроения ФГБОУ ВО «МИРЭА – Российский технологический университет» (119454, Россия, Москва, пр-т Вернадского, д. 78).

About the authors:

Shamil G. Magomedov, Cand of Sci. (Engineering), Associate Professor of the Chair CS-4 “Automated Control Systems”, Institute of Integrated Security and Special Instrumentation, MIREA – Russian Technological University (78, Vernadskogo pr., Moscow 119454, Russia).

Artem S. Lebedev, Lecturer of the Chair CS-4 “Automated Control Systems”, Institute of Integrated Security and Special Instrumentation, MIREA – Russian Technological University (78, Vernadskogo pr., Moscow 119454, Russia).