А. С. Лебедев

Организация информационного обмена между параллельными процессами при автоматическом распараллеливании линейных программ для кластерных систем с применением модели многогранников

Аннотация. В работе предложен метод организации информационного обмена между параллельными процессами для преобразования программы, сгенерированной инструментом CLooG на основе расписания и размещения вычислений, в параллельную MPI-программу, учитывающую оптимальное размещение данных. Предполагается, что пространственно-временные преобразования программы являются аффинными и вычисляются методами модели многогранников в соответствии с принципом оптимизации временной и пространственной локальности данных. Введено понятие многогранника коммуникаций для определения набора данных, участвующих в информационном обмене с двухсторонней коммуникацией процессов. Применение метода проиллюстрировано на примере распараллеливания алгоритма LU-разложения.

Ключевые слова и фразы: Автоматическое распараллеливание, модель многогранников, локальность данных, кластерные системы, информационный обмен.

Введение

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

Исследование выполнено при финансовой поддержке РФФИ в рамках проекта 16-07-00096 А.

© А. С. Лебедев, 2017

© Институт системного анализа ФИЦ «Информатика и управление» РАН, 2017

© Программные системы: теория и приложения, 2017

DOI: 10.25209/2079-3316-2017-8-4-3-20

Большинство вычислительно емких научных и инженерных приложений тратят значительную часть времени исполнения на вложенности циклов, часто отвечающие критериям линейности программ. Модель многогранников [1] предоставляет мощный математический аппарат, упрощающий анализ и преобразование таких программ с целью улучшения быстродействия путем повышения степени параллелизма и оптимизации локальности данных при вычислениях. Методы модели многогранников широко используются в статической компиляции [2–4]. Код, написанный на языке высокого уровня (чаще всего это C или FORTRAN), оптимизируется для параллельного выполнения и компилируется для целевой архитектуры. Распараллеливание в модели многогранников осуществляется в пять этапов.

- (1) Анализ информационных зависимостей [5]. Рассматриваются потоковые зависимости (чтение после записи), антивисимости (запись после чтения), зависимости через выход (запись после записи).
- (2) Построение многомерного аффинного расписания программы, обладающего параллелизмом и сохраняющего логический порядок операций в соответствии с зависимостями по данным. Широко применяемыми на практике стали классические методы поиска расписаний, ориентированные на минимизацию задержки расписания [6, 7], и современный подход [4], ориентированный на оптимизацию локальности данных.
- (3) Построение пространственного размещения вычислений и данных. Вычисляется аффинное размещение, обладающее параллелизмом, и уменьшающее стоимость коммуникаций [8, 9].
- (4) Разбиение индексного пространства виртуальных процессоров [9, 10] — сопоставление виртуальных процессоров физическим.
- (5) Генерация параллельной программы по алгоритму CLooG [11] в соответствии с пространственно-временным преобразованием, заданным функциями расписания и размещения, и разбиением.

Ранее автором были проанализированы современные подходы к вычислению пространственно-временных преобразований, и предложен новый, обладающий более гибкими средствами задания критериев качества альтернатив, подход [12], реализующий принцип оптимизации временной и пространственной локальности данных [4] с расширением жадной схемы Футриера [7] при вычислении многомерных расписаний и сохранением свойства вперед направленных коммуникаций (forward communication only placements, FCO) [9] при

вычислении размещений. Предложенный в [12] автором метод применим для вычислительных систем SMP и NUMA, и предназначен для конструирования параллельных программ с помощью генератора CLoog [11] и директив OpenMP (`#pragma omp parallel for`), размещаемых перед заголовками параллельных циклов (циклов, итерирующих по пространству виртуальных процессоров). Позднее был предложен метод построения размещений вычислений и данных для систем с распределенной памятью (кластеров) [13], также сохраняющий свойство FCO и реализующий принцип пространственной локальности данных [4]. Если задача кодогенерации для систем с общей памятью решается механически применением кодогенератора CLoog и расстановкой директив OpenMP, то конструирование параллельной программы для систем с распределенной памятью требует организации информационного обмена между параллельными процессами. Существующие работы в этом направлении предполагают трансляцию произвольной OpenMP-программы в MPI-программу и не оптимизируют пространственную локальность данных: STEP [14] предполагает, что данные загружены на всех процессорах, OMP2MPI [15] вводит понятие главного процесса, являющегося планировщиком работ и брокером данных с общим доступом.

Целью настоящей работы является разработка метода организации информационного обмена между параллельными процессами для преобразования программы, сгенерированной CLoog [11] на основе расписания и размещения вычислений, в параллельную MPI-программу, учитывающую оптимальное размещение данных [13].

В первой части приводится обзор основных понятий модели многогранников, во второй части следует описание разработанного метода для организации информационного обмена между процессами. В третьей части приводится пример распараллеливания программы LU-разложения и сравнение быстродействия с последовательным вариантом программы.

1. Базовые понятия модели многогранников

Модель многогранников — модель последовательных и параллельных вычислений, рассматривающая класс линейных программ [16]. Эти программы удовлетворяют следующим ограничениям:

- единственный тип исполнительного оператора — оператор присваивания, правая часть которого является арифметическим выражением;

- все повторяющиеся операции описываются только с помощью циклов DO языка FORTRAN (либо эквивалентными циклами for языка C); структура вложенности циклов может быть произвольной; шаги изменения параметров циклов всегда равны +1;
- допускается использование условных и безусловных операторов перехода, передающих управление вниз по тексту; не допускаются побочные выходы из циклов;
- все индексные выражения переменных, границы изменения параметров циклов и условия передачи управления задаются однородными формами, линейными по совокупности параметров циклов и внешних переменных программы; все коэффициенты линейных форм являются целыми числами;
- внешние переменные программы всегда целочисленные; конкретные значения внешних переменных известны только во время работы программы.

Обобщенный граф зависимостей — это ориентированный мультиграф, каждая вершина которого представляет множество соответственных операций (динамических экземпляров одной и той же инструкции). Каждая операция принадлежит только одной вершине. Ребро графа представляет временное ограничение на две операции, соответствующие начальной и конечной вершине (принцип причинности для информационно зависимых операций u и v : $u \rightarrow v \Rightarrow \theta(v) > \theta(u)$, где θ — логическое время). В обобщенном графе зависимостей могут быть петли и циклы. Каждая вершина помечается описанием соответствующих операций, а каждое ребро — описанием соответствующих отношений зависимости.

Каждой вершине X соответствует ее домен — многогранник D_X на множестве $\mathbb{Q}^{p_X}$, 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. Количество внешних переменных обозначим q_z .

Каждому ребру e , исходящему из вершины $\sigma(e) = X$ в $\delta(e) = Y$, ставится в соответствие многогранник R_e на множестве $\mathbb{Q}^{p_X + p_Y}$ такой, что условие причинности должно быть наложено на операции $\langle \vec{i}, \vec{z}; X \rangle$ и $\langle \vec{j}, \vec{z}; Y \rangle$ тогда и только тогда, когда составной вектор $\begin{bmatrix} \vec{i} \\ \vec{j} \end{bmatrix} \in R_e$.

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

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

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

Тем самым d -мерное ($d \geq 1$) расписание для обобщенного графа зависимостей есть функция $\theta: \Omega \rightarrow \mathbb{N}_0^d$ такая, что $\forall e \in E, \vec{x} \in D_{\sigma(e)}, \vec{y} \in D_{\delta(e)}$

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

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

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

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

```

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

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

Размещение для обобщенного графа зависимостей есть функция $\pi: \Omega \rightarrow \mathbb{N}_0$, которая ставит в соответствие операции номер виртуального процессора, на котором она будет исполняться. По аналогии вводится понятие размещения для массивов данных. Обозначим $\langle g_a, \vec{z}; A_a \rangle$ доступ к массиву A_a с аффинной индексной функцией $g_a(\vec{i})$, $\vec{i} \in D_X$ в некоторой позиции a инструкции X . Тогда $\eta(\langle g_a, \vec{z}; A_a \rangle)$ — номер виртуального процессора, на котором располагается элемент $A_a[g_a(\vec{i})]$, $\vec{i} \in D_X$. В дальнейшем будем обозначать p_A — размерность

массива A , D_A — множество допустимых индексов (прямоугольный параллелепипед в $\mathbb{Z}^{p_A}$).

Одномерное аффинное размещение для инструкции X есть функция вида $\pi_X(\vec{i}, \vec{z}) = \vec{v}_X \cdot \vec{i} + \vec{v}'_X \cdot \vec{z} + v^0_X$, $\vec{v}_X \in \mathbb{Z}^{p_X}$, $\vec{v}'_X \in \mathbb{Z}^{q_z}$, $v^0_X \in \mathbb{Z}$, $\vec{i} \in D_X$.

Одномерное аффинное размещение для массива A есть функция вида $\eta_A(\vec{g}, \vec{z}) = \vec{v}_A \cdot \vec{g} + \vec{v}'_A \cdot \vec{z} + v^0_A$, $\vec{v}_A \in \mathbb{Z}^{p_A}$, $\vec{v}'_A \in \mathbb{Z}^{q_z}$, $v^0_A \in \mathbb{Z}$, $\vec{g} \in D_A$.

2. Генерация кода для систем с распределенной памятью

Рассматривается параллельная программа, полученная по методу [13]. Пространство виртуальных процессоров одномерное. Программа содержит l массивов A_k , $k = 1, \dots, l$, m инструкций S_i , $i = 1, \dots, m$, n обращений к памяти a_j , $j = 1, \dots, n$.

На практике часто применяется равномерное распределение виртуальных процессоров между физическими. Пусть вычислительная система имеет Q физических процессоров, проиндексированных переменной $r = 0, 1, \dots, Q - 1$. Каждый физический процессор r обрабатывает отрезок пространства виртуальных процессоров $l(r), \dots, u(r)$:

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

В этом случае не все Q физических процессоров могут быть нагружены, но количество виртуальных процессоров, сопоставленных самому нагруженному физическому процессору, является минимальным из возможных.

Информационный обмен между физическими процессорами обуславливается операциями удаленного чтения и удаленной записи, когда процессор с индексом R осуществляет доступ к элементам данных, расположенным на процессоре с индексом r : $R > r$ для операций удаленного чтения, $R < r$ для операций удаленной записи, поскольку построенные размещения вычислений и данных удовлетворяют свойству вперед направленных коммуникаций [9].

Будем строить программу с синхронным параллелизмом по следующей схеме:

```

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

Перед выполнением фронта происходит пересылка данных для операций удаленного чтения, после выполнения фронта происходит пересылка данных, индуцированных операциями удаленной записи. Реализация параллелизма предполагается в рамках стандарта MPI с двухсторонней коммуникацией процессов (MPI_Send, MPI_Recv). Определим множество данных, которые будут участвовать в актах информационного обмена при условии, что разбиение пространства виртуальных процессоров будет выполняться по схеме равномерного распределения.

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

Обозначим C — контекст [11] — многогранник внешних переменных программы, заданный p_C ограничениями вида $\vec{c}_k \cdot \vec{z} + d_k \geq 0$, $\vec{c}_k \in \mathbb{Z}^{q_z}$, $d_k \in \mathbb{Z}$, $k = 1, \dots, p_C$.

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

ОПРЕДЕЛЕНИЕ 1. Мгновенным доменом инструкции S для физического процессора R назовем параметрически заданный многогранник $D''_{S,R}$:

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

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

Контекст $C''_{S,R}$, определяющий параметры $D''_{S,R}$, задается следующей системой ограничений:

$$\begin{cases} \vec{c}_k \cdot \vec{z} + d_k \geq 0, k = 1, \dots, p_C, \\ L \leq l(R) \leq u(R) \leq U, \\ \vec{b}_{S,k} \cdot \vec{i}'_S + \vec{c}_{S,k} \cdot \vec{z} + d_{S,k} \geq 0, k \in K^{C''}_S. \end{cases}$$

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

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

Индексы $\vec{g}_{A_a}$ элементов массива A_a , которые процессор R должен принять от процессора r перед выполнением фронта, в котором происходит удаленное чтение в позиции a инструкции S , удовлетворяют условиям

$$Q_a^{r-r}(\vec{g}_{A_a}, R, r) : \begin{cases} r < R, \\ \vec{g}_{A_a} \in \tilde{g}_a(\theta', D''_{S,R}), \\ l(r) \leq \eta(\langle \vec{g}_{A_a}, \vec{z}; A_a \rangle) \leq u(r). \end{cases}$$

Индексы $\vec{g}_{A_a}$ элементов массива A_a , которые процессор R должен отправить процессору r перед выполнением фронта, в котором происходит удаленное чтение в позиции a инструкции S , удовлетворяют условиям

$$Q_a^{r-s}(\vec{g}_{A_a}, R, r) : \begin{cases} r > R, \\ \vec{g}_{A_a} \in \tilde{g}_a(\theta', D''_{S,r}), \\ l(R) \leq \eta(\langle \vec{g}_{A_a}, \vec{z}; A_a \rangle) \leq u(R). \end{cases}$$

Индексы $\vec{g}_{A_a}$ элементов массива A_a , которые процессор R должен отправить процессору r после выполнения фронта, в котором происходит удаленная запись в позиции a инструкции S , удовлетворяют

УСЛОВИЯМ

$$Q_a^{w-s}(\vec{g}_{A_a}, R, r) : \begin{cases} r > R, \\ \vec{g}_{A_a} \in \tilde{g}_a(\theta', D''_{S,R}), \\ l(r) \leq \eta(\langle \vec{g}_{A_a}, \vec{z}; A_a \rangle) \leq u(r). \end{cases}$$

Индексы $\vec{g}_{A_a}$ элементов массива A_a , которые процессор R должен принять от процессора r после выполнения фронта, в котором происходит удаленная запись в позиции a инструкции S , удовлетворяют условиям

$$Q_a^{w-r}(\vec{g}_{A_a}, R, r) : \begin{cases} r < R, \\ \vec{g}_{A_a} \in \tilde{g}_a(\theta', D''_{S,r}), \\ l(R) \leq \eta(\langle \vec{g}_{A_a}, \vec{z}; A_a \rangle) \leq u(R). \end{cases}$$

ОПРЕДЕЛЕНИЕ 2. Множества $Q_*^{r/w-r/s}$ являются параметрически заданными многогранниками. Назовем их многогранниками коммуникаций. В первой части верхнего индекса r означает операцию удаленного чтения (*read*), w — операцию удаленной записи (*write*). Во второй части верхнего индекса r означает прием данных (*receive*), s — отправку (*send*).

Такие многогранники могут быть просканированы вложенностями циклов, для генерации которых подходит инструмент CLooG [11]. Код этих вложенностей можно использовать при формировании пакетов для осуществления информационного обмена между процессами в MPI-коммуникаторе.

Уточним шаг 2 параллельной программы:

```
// exchange data for remote reads
for (int j = 1; j <= n; j++) {
  if (a_j is read) {
    for (int r = 0; r < R; r++)
      MPI_Recv(data for Q_{a_j}^{r-r}(\vec{g}_{A_{a_j}}, R, r));
    for (int r = R + 1; r < Q; r++)
      MPI_Send(data for Q_{a_j}^{r-s}(\vec{g}_{A_{a_j}}, R, r));
  }
}
```

Уточним шаг 5 параллельной программы:

```
// exchange data for remote writes
for (int j = 1; j <= n; j++) {
  if (a_j is write) {
```

```

for (int r = 0; r < R; r++)
    MPI_Recv(data for  $Q_{a_j}^{w-r}(\bar{g}_{Aa_j}, R, r)$ );
for (int r = R + 1; r < Q; r++)
    MPI_Send(data for  $Q_{a_j}^{w-s}(\bar{g}_{Aa_j}, R, r)$ );
}

```

Переменная R хранит значение собственного индекса физического процессора (ранг MPI-процесса). Переменная r используется для индексирования других физических процессоров (рангов других MPI-процессов).

3. Пример: LU-разложение

Рассмотрим алгоритм LU-разложения квадратной матрицы A .

```

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

```

Применение методов [12] и [13] дает следующие аффинные преобразования: $\theta_{S_1} = 2k$, $\theta_{S_2} = 2k + 1$, $\pi_{S_1} = l$, $\pi_{S_2} = i$, $\eta_A = \bar{g}_A^{(1)}$.

Cloog [11] генерирует следующую вложенность циклов для приведенных пространственно-временных преобразований:

```

#define ceild(n,d) ceil(((double)(n))/((double)(d)))

for (int c1=0; c1<=2*N-3; c1++) {
    // Parallel loop
    for (int c2=ceild(c1+1,2); c2<=N-1; c2++) {
        if ((c1+1)%2==0) {
            for (int c8=ceild(c1+1,2); c8<=N-1; c8++)
                //S1  $r_1, w_1$   $r_2$   $r_3$ 
                A[c2][c8] -= A[c2][ (c1-1)/2 ] * A[ (c1-1)/2 ][c8];
        }
        if (c1%2==0) {
            //S2  $r_4, w_2$   $r_5$ 
            A[c2][c1/2] = A[c1/2][c1/2];
        }
    }
}

```

В комментариях отмечено, что что цикл по c_2 является параллельным (итерирует по пространству виртуальных процессоров), а также заново пронумерованы инструкции и позиции, в которых осуществляется доступ к памяти.

3.1. Анализ программы

Цикл по c_2 содержит фронт операций. Выпишем многогранники коммуникаций:

$$\begin{aligned}
 C''_{S_1, R} : \begin{cases} N \geq 0, \\ L \leq l(R) \leq u(R) \leq U, \\ c_1 \geq 0, \\ c_1 \leq 2N - 3, \\ c_1 = 2v + 1, \\ v \geq 0; \end{cases} & D''_{S_1, R} : \begin{cases} l(R) \leq c_2 \leq u(R), \\ 2c_8 \geq c_1 + 1, \\ c_8 \leq N - 1; \end{cases} \\
 \vec{g}_{A_{r_1, w_1}} = \begin{bmatrix} c_2 \\ c_8 \end{bmatrix}; \quad \tilde{g}_{r_1, w_1}([c_1], D''_{S_1, R}) : \begin{cases} l(R) \leq \vec{g}_{A_{r_1, w_1}}^{(1)} \leq u(R), \\ 2g_{A_{r_1, w_1}}^{(2)} \geq c_1 + 1, \\ g_{A_{r_1, w_1}}^{(2)} \leq N - 1; \end{cases} \\
 \vec{g}_{A_{r_2}} = \begin{bmatrix} c_2 \\ (c_1 - 1)/2 \end{bmatrix}; \quad \tilde{g}_{r_2}([c_1], D''_{S_1, R}) : \begin{cases} l(R) \leq \vec{g}_{A_{r_2}}^{(1)} \leq u(R), \\ \vec{g}_{A_{r_2}}^{(2)} = v; \end{cases} \\
 \vec{g}_{A_{r_3}} = \begin{bmatrix} (c_1 - 1)/2 \\ c_8 \end{bmatrix}; \quad \tilde{g}_{r_3}([c_1], D''_{S_1, R}) : \begin{cases} \vec{g}_{A_{r_3}}^{(1)} = v, \\ \vec{g}_{A_{r_3}}^{(2)} \geq v + 1, \\ \vec{g}_{A_{r_3}}^{(2)} \leq N - 1; \end{cases} \\
 C''_{S_2, R} : \begin{cases} N \geq 0, \\ L \leq l(R) \leq u(R) \leq U, \\ c_1 \geq 0, \\ c_1 \leq 2N - 3, \\ c_1 = 2v, \\ v \geq 0; \end{cases} & D''_{S_2, R} : l(R) \leq c_2 \leq u(R); \\
 \vec{g}_{A_{r_4}} = \begin{bmatrix} c_2 \\ c_1/2 \end{bmatrix}; \quad \tilde{g}_{r_4}([c_1], D''_{S_2, R}) : \begin{cases} l(R) \leq g_{A_{r_4}}^{(1)} \leq u(R), \\ \vec{g}_{A_{r_4}}^{(2)} = v; \end{cases} \\
 \vec{g}_{A_{r_5}} = \begin{bmatrix} c_1/2 \\ c_1/2 \end{bmatrix}; \quad \tilde{g}_{r_5}([c_1], D''_{S_2, R}) : \begin{cases} \vec{g}_{A_{r_5}}^{(1)} = v, \\ \vec{g}_{A_{r_5}}^{(2)} = v. \end{cases}
 \end{aligned}$$

В позиции r_3 находится операция удаленного чтения. Процессор, на котором расположена v -я строка матрицы, должен отправить ее фрагмент $A[v][v + 1 : N - 1]$ всем остальным процессорам с большим индексом перед исполнением фронта:

$$Q_{r_3}^{r-r}(\vec{g}_{A_{r_3}}, R, r) : \begin{cases} r < R, \\ \vec{g}_{A_{r_3}}^{(1)} = v, \\ \vec{g}_{A_{r_3}}^{(2)} \geq v + 1, \\ \vec{g}_{A_{r_3}}^{(2)} \leq N - 1, \\ l(r) \leq g_{A_{r_3}}^{(1)} \leq u(r), \end{cases}$$

$$Q_{r_3}^{r-s}(\vec{g}_{A_{r_3}}, R, r) : \begin{cases} r > R, \\ \vec{g}_{A_{r_3}}^{(1)} = v, \\ \vec{g}_{A_{r_3}}^{(2)} \geq v + 1, \\ \vec{g}_{A_{r_3}}^{(2)} \leq N - 1, \\ l(R) \leq g_{A_{r_3}}^{(1)} \leq u(R). \end{cases}$$

В позиции r_5 также находится операция удаленного чтения. Процессор, на котором расположен элемент матрицы $A[v][v]$, должен отправить его всем остальным процессорам с большим индексом перед исполнением фронта:

$$Q_{r_5}^{r-r}(\vec{g}_{A_{r_5}}, R, r) : \begin{cases} r < R, \\ \vec{g}_{A_{r_5}}^{(1)} = v, \\ \vec{g}_{A_{r_5}}^{(2)} = v, \\ l(r) \leq g_{A_{r_5}}^{(1)} \leq u(r), \end{cases}$$

$$Q_{r_5}^{r-s}(\vec{g}_{A_{r_5}}, R, r) : \begin{cases} r > R, \\ \vec{g}_{A_{r_5}}^{(1)} = v, \\ \vec{g}_{A_{r_5}}^{(2)} = v, \\ l(R) \leq g_{A_{r_5}}^{(1)} \leq u(R). \end{cases}$$

В остальных позициях нет операций удаленного доступа, и соответственные многогранники коммуникаций — пустые множества.

Следует отметить, что при четном c_1 контекст $C''_{S_1, R}$ является пустым, инструкция S_1 не выполняется, и для нее не нужно строить многогранники коммуникаций. Аналогично, при нечетном c_1 контекст

$C''_{S_2,R}$ является пустым, инструкция S_2 не выполняется, и для нее также не нужно строить многогранники коммуникаций.

Следующая MPI-программа осуществляет информационный обмен в соответствии с найденными многогранниками коммуникаций:

```
int numproc, id;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &numproc);
MPI_Comm_rank(MPI_COMM_WORLD, &id);

int chunk_size = (N + numproc - 1) / numproc;
int actual_chunk_size = (id == numproc - 1) ?
    N - (numproc - 1) * chunk_size : chunk_size;

size_t sz1 = sizeof(double*) * actual_chunk_size;
size_t sz2 = sizeof(double) * N;

double** A = (double**) malloc(sz1);
for (int row = 0; row < actual_chunk_size; row++)
    A[row] = (double*) (malloc(sz2));

for (int c1 = 0; c1 <= 2 * N - 3; c1++) {
    int row_idx;
    int row_length;

    if ((c1 + 1) % 2 == 0) {
        row_idx = (c1-1)/2;
        row_length = N - row_idx;
    }

    if (c1 % 2 == 0) {
        row_idx = c1/2;
        row_length = 1;
    }

    double* row_ptr = NULL;

    int owner_id = row_idx / chunk_size;

    // send the line #row_idx owned by me to all the row_idx+1..N-1
    // virtual processors or id+1..numproc mpi processes
    if (id == owner_id) {
        row_ptr = A[row_idx % chunk_size];
        for (int addressee = id + 1; addressee < numproc; addressee++)
            MPI_Send(row_ptr + row_idx, row_length, MPI_DOUBLE, addressee, 0,
                MPI_COMM_WORLD);
    }
    else if (id > owner_id) {
        row_ptr = got_row;
        MPI_Status status;
        MPI_Recv(row_ptr + row_idx, row_length, MPI_DOUBLE, owner_id,
            MPI_ANY_TAG, MPI_COMM_WORLD, &status);
    }
}
```

Рис. 1. Ускорение от распараллеливания LU-разложения

```

}

MPI_Barrier(MPI_COMM_WORLD);

for (int c2 = max(id * chunk_size, ceilid(c1 + 1, 2));
     c2 <= id * chunk_size + actual_chunk_size - 1; c2++) {
    if ((c1 + 1) % 2 == 0) {
        for (int c8 = ceilid(c1 + 1, 2); c8 <= N - 1; c8++)
            A[c2 % chunk_size][c8] -= A[c2 % chunk_size][(c1-1)/2]*row_ptr[c8];
    }
    if (c1 % 2 == 0) {
        A[c2 % chunk_size][c1/2] /= row_ptr[c1/2];
    }
}

MPI_Barrier(MPI_COMM_WORLD);
}
//...

```

3.2. Вычислительные эксперименты

Эксперименты проводились для квадратной матрицы порядка $N = 2048$ вещественных чисел двойной точности на узлах кластера РГАТУ [17]. Использовалась библиотека OpenMPI и единственный процессор на каждом узле. Индексы виртуальных процессоров равномерно распределены между физическими. На рис. 1 показана зависимость коэффициента ускорения вычислений от количества узлов. Программа показала приемлемую масштабируемость на пяти задействованных узлах — выигрыш от распараллеливания составил 2,75 раза.

Заключение

Метод организации информационного обмена между параллельными процессами, предложенный в статье, позволяет преобразовать

программу, сгенерированную инструментом CLoG на основе расписания и размещения вычислений, в параллельную MPI-программу для кластерных систем, учитывающую оптимальное размещение данных. Предполагается, что параллельное расписание найдено методом [12], а оптимизация размещения вычислений и данных произведена методом [13], обеспечивающим свойство вперед направленных коммуникаций. Это свойство гарантирует отсутствие тупиков (deadlocks) при выполнении информационного обмена для операций удаленного чтения и удаленной записи. Метод успешно применен для преобразования и запуска программы LU-разложения в MPI-среде с двухсторонней коммуникацией процессов.

В рассмотренном примере многогранники коммуникаций определяют данные, расположенные в непрерывном участке памяти, что позволяет непосредственно вызвать `MPI_Send()` и `MPI_Recv()` для обеспечения информационного обмена. В общем случае многогранники коммуникаций могут иметь более сложную форму и определять данные, занимающие несколько непрерывных областей памяти. Техники упаковки данных в информационные пакеты и последующей их распаковки являются предметом дальнейших исследований.

Список литературы

- [1] Ch. Lengauer. “Loop parallelization in the polytope model”, International Conference on Concurrency Theory CONCUR’93 (Hildesheim, Germany, August 23–26, 1993), Lecture Notes in Computer Science, vol. **715**, Springer, Berlin–Heidelberg, 1993, pp. 398–416. [↑] ⁴
- [2] F. Irigoin, P. Jouvelot, R. Triolet. “Semantical interprocedural parallelization: An overview of the PIPS project”, *ACM International Conference on Supercomputing 25th Anniversary Volume* (Munich, Germany, June 10–13, 2014), ACM, 2014, pp. 143–150. [↑] ⁴
- [3] M. Griebel, Ch. Lengauer. “The loop parallelizer LooPo”, *Proc. Sixth Workshop on Compilers for Parallel Computers*, Konferenzen des Forschungszentrums Jülich, vol. **21**, 1996, pp. 311–320. [↑] ⁴
- [4] U. Bondhugula, M. Baskaran, S. Krishnamoorthy, J. Ramanujam, A. Rountev, P. Sadayappan. “Automatic transformations for communication-minimized parallelization and locality optimization in the polyhedral model”, International Conference on Compiler Construction CC 2008 (Budapest, Hungary, March 29–April 6, 2008), Lecture Notes in Computer Science, vol. **4959**, Springer, Berlin–Heidelberg, 2008, pp. 132–146. [↑] ^{4,5}
- [5] P. Feautrier. “Dataflow analysis of array and scalar references”, *International Journal of Parallel Programming*, **20**:1 (1991), pp. 23–53. [↑] ^{4,7}
- [6] P. Feautrier. “Some efficient solutions to the affine scheduling problem. I. One-dimensional time”, *International Journal of Parallel Programming*, **21**:5 (1992), pp. 313–347. [↑] ⁴

- [7] P. Feautrier. “Some efficient solutions to the affine scheduling problem. Part II. Multidimensional time”, *International Journal of Parallel Programming*, **21**:6 (1992), pp. 389–420. [↑] [4](#)
- [8] P. Feautrier. “Toward automatic distribution”, *Parallel Processing Letters*, **4**:03 (1994), pp. 233–244. [↑] [4](#)
- [9] M. Griebl. *Automatic parallelization of loop programs for distributed memory architectures*, Habilitation, Univ. Passau, 2004, 207 p., URL: <http://www.infosun.fim.uni-passau.de/cl/publications/docs/Gri04.pdf> [↑] [4,8](#)
- [10] V. Bandishti, I. Pananilath, U. Bondhugula. “Tiling stencil computations to maximize parallelism”, *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC ’12 (Salt Lake City, Utah, USA, November 10–16, 2012), IEEE Computer Society Press, 2012, 11 p. [↑] [4](#)
- [11] C. Bastoul. “Code generation in the polyhedral model is easier than you think”, *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*, PACT’04 (Antibes Juan-les-Pins, France, 29 September–3 October 2004), IEEE Computer Society, 2004, pp. 7–16, URL: <https://hal.archives-ouvertes.fr/hal-00017260> [↑] [4,5,9,11,12](#)
- [12] А. С. Лебедев. «Пространственно-временные преобразования при распараллеливании линейных программ», *Информационные технологии и вычислительные системы*, 2015, №1, с. 19–32. [↑] [4,5,12,17](#)
- [13] А. С. Лебедев. «Размещение данных при автоматическом распараллеливании линейных программ для систем с распределенной памятью», *Вестник Рыбинской государственной авиационной технологической академии им. П. А. Соловьева*, 2015, №3, с. 92–99. [↑] [5,8,12,17](#)
- [14] D. Millot, A. Muller, Ch. Parrot, F. Silber-Chaussumier. “STEP: a distributed OpenMP for coarse-grain parallelism tool”, IWOMP 2008 (West Lafayette, IN, USA, May 12–14, 2008), Lecture Notes in Computer Science, vol. **5004**, 2008, pp. 83–99. [↑] [5](#)
- [15] A. Saà-Garriga, D. Castells-Rufas, J. Carrabina. *OMP2MPI: Automatic MPI code generation from OpenMP programs*, 2015, URL: <https://arxiv.org/abs/1502.02921> [↑] [5](#)
- [16] В. В. Воеводин. *Параллельные вычисления*, БХВ-Петербург, СПб., 2004. [↑] [5](#)
- [17] Кластер IBM ПГАТУ, URL: <http://kbit.rsatu.ru/ibmcluster/> [↑] [16](#)

Рекомендовал к публикации

д.т.н. В. М. Хачумов

Пример ссылки на эту публикацию:

А. С. Лебедев. «Организация информационного обмена между параллельными процессами при автоматическом распараллеливании линейных программ для кластерных систем с применением модели многогранников», *Программные системы: теория и приложения*, 2017, **8**:4(35), с. 3–20.

URL: http://psta.psisras.ru/read/psta2017_4_3-20.pdf

Об авторе:

Артём Сергеевич Лебедев

Младший научный сотрудник ИСА ФИЦ ИУ РАН. Область научных интересов: параллельное программирование, автоматическое распараллеливание программ, вычислительная математика, распределенные приложения, трансляторы.

e-mail:

tementy@gmail.com

Artem Lebedev. *Organizing communication of parallel processes during automatic parallelization of loop nests with static control flow for cluster systems using polyhedral model.*

ABSTRACT. A method to organize communication of parallel processes is proposed for translation of CLoOG output into parallel MPI program preserving optimal data placement. It is assumed that space-time mappings are affine and calculated using polyhedral model methods according to the principle of space and time data locality optimization. The notion of a communication polyhedron is introduced to define a data set participating in any information exchange within two-sided MPI communication scheme. Application of the method is illustrated with parallelization of LU-decomposition algorithm. (*in Russian*).

Key words and phrases: Automatic parallelization, polyhedral model, data locality, clusters, intercommunications.

References

- [1] Ch. Lengauer. “Loop parallelization in the polytope model”, International Conference on Concurrency Theory CONCUR’93 (Hildesheim, Germany, August 23–26, 1993), Lecture Notes in Computer Science, vol. **715**, Springer, Berlin–Heidelberg, 1993, pp. 398–416.
- [2] F. Irigoin, P. Jouvelot, R. Triolet. “Semantical interprocedural parallelization: An overview of the PIPS project”, *ACM International Conference on Supercomputing 25th Anniversary Volume* (Munich, Germany, June 10–13, 2014), ACM, 2014, pp. 143–150.
- [3] M. Griebl, Ch. Lengauer. “The loop parallelizer LooPo”, *Proc. Sixth Workshop on Compilers for Parallel Computers*, Konferenzen des Forschungszentrums Jülich, vol. **21**, 1996, pp. 311–320.
- [4] U. Bondhugula, M. Baskaran, S. Krishnamoorthy, J. Ramanujam, A. Rountev, P. Sadayappan. “Automatic transformations for communication-minimized parallelization and locality optimization in the polyhedral model”, International Conference on Compiler Construction CC 2008 (Budapest, Hungary, March 29–April 6, 2008), Lecture Notes in Computer Science, vol. **4959**, Springer, Berlin–Heidelberg, 2008, pp. 132–146.

- [5] P. Feautrier. “Dataflow analysis of array and scalar references”, *International Journal of Parallel Programming*, **20**:1 (1991), pp. 23–53.
- [6] P. Feautrier. “Some efficient solutions to the affine scheduling problem. I. One-dimensional time”, *International Journal of Parallel Programming*, **21**:5 (1992), pp. 313–347.
- [7] P. Feautrier. “Some efficient solutions to the affine scheduling problem. Part II. Multidimensional time”, *International Journal of Parallel Programming*, **21**:6 (1992), pp. 389–420.
- [8] P. Feautrier. “Toward automatic distribution”, *Parallel Processing Letters*, **4**:03 (1994), pp. 233–244.
- [9] M. Griebl. *Automatic parallelization of loop programs for distributed memory architectures*, Habilitation, Univ. Passau, 2004, 207 p., URL: <http://www.infosun.fim.uni-passau.de/cl/publications/docs/Gri04.pdf>
- [10] V. Bandishti, I. Pananilath, U. Bondhugula. “Tiling stencil computations to maximize parallelism”, *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC ’12 (Salt Lake City, Utah, USA, November 10–16, 2012), IEEE Computer Society Press, 2012, 11 p.
- [11] C. Bastoul. “Code generation in the polyhedral model is easier than you think”, *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*, PACT’04 (Antibes Juan-les-Pins, France, 29 September–3 October 2004), IEEE Computer Society, 2004, pp. 7–16, URL: <https://hal.archives-ouvertes.fr/hal-00017260>
- [12] A. S. Lebedev. “Space-time mappings for parallelization of affine programs”, *Informatsionnyye tekhnologii i vychislitel’nyye sistemy*, 2015, no.1, pp. 19–32.
- [13] A. S. Lebedev. “Construction of data placements for automatic parallelization of affine programs for distributed memory systems”, *Vestnik Rybinskoy gosudarstvennoy aviatsionnoy tekhnologicheskoy akademii im. P. A. Solov’yeva*, 2015, no.3, pp. 92–99.
- [14] D. Millot, A. Muller, Ch. Parrot, F. Silber-Chaussumier. “STEP: a distributed OpenMP for coarse-grain parallelism tool”, IWOMP 2008 (West Lafayette, IN, USA, May 12–14, 2008), *Lecture Notes in Computer Science*, vol. **5004**, 2008, pp. 83–99.
- [15] A. Saà-Garriga, D. Castells-Rufas, J. Carrabina. *OMP2MPI: Automatic MPI code generation from OpenMP programs*, 2015, URL: <https://arxiv.org/abs/1502.02921>
- [16] V. V. Voevodin. *Parallel computing*, BKhV-Peterburg, SPb., 2004.
- [17] Cluster IBM 1350, URL: <http://kbit.rsatu.ru/ibmcluster/>

Sample citation of this publication:

Artem Lebedev. “Organizing communication of parallel processes during automatic parallelization of loop nests with static control flow for cluster systems using polyhedral model”, *Program systems: Theory and applications*, 2017, **8**:4(35), pp. 3–20. (In Russian).

URL: http://psta.psiras.ru/read/psta2017_4.3-20.pdf