УДК 004.4’422

НА ПУТИ К СОЗДАНИЮ РАСПАРАЛЛЕЛИВАЮЩИХ КОМПИЛЯТОРОВ НА ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ С РАСПРЕДЕЛЕННОЙ ПАМЯТЬЮ

Б. Я. Штейнберг[0000-0001-8146-0479]

Южный федеральный университет,
Институт математики, механики и компьютерных наук

byshtyaynberg@sfedu.ru

Аннотация

Описаны условия создания оптимизирующих распараллеливающих компиляторов на вычислительные системы с распределённой памятью. Целевые вычислительные системы – это микросхемы типа «суперкомпьютер на кристалле». Приведены как оптимизирующие преобразования программ специфичные для систем с распределенной памятью, так и такие преобразования, которые нужны и для вычислительных систем с распределенной памятью, и для вычислительных систем с общей памятью. Обсуждены вопросы минимизации межпроцессорных пересылок при распараллеливании рекурсивной функции. Основной подход к созданию таких компиляторов – блочно-аффинные размещения данных в распределенной памяти с минимизацией межпроцессорных пересылок. Показано, что создавать распараллеливающие компиляторы на вычислительные системы с распределенной памятью следует на основе высокоуровневого внутреннего представления и с высокоуровневым выходным языком.

Ключевые слова*: автоматизация распараллеливания, распределенная память, преобразования программ, размещение данных, пересылки данных*

ВВЕДЕНИЕ

Во многих публикациях для создания параллельного кода для систем с распределенной памятью предложены полуавтоматические инструменты, в которых пользователь должен вызывать специальные функции или писать директивы компилятору. В работе [1] отмечено, что автоматическая компиляция последовательной программы для параллельной архитектуры с распределенной памятью

является очень сложной задачей, для которой нет в настоящее время эффективного решения. В этой же работе описана генерация параллельного кода, основанного на MPI (для кластера с мультипроцессорами). Об оптимизации размещения данных не говорится ничего. Использована распараллеливающая система Pluto, которая позволяет в пространстве итераций находить подмножества точек, допускающих параллельное выполнение. Ничего не говорится о локализации данных, которая необходима для эффективности на современных вычислительных архитектурах при параллельном выполнении относительно больших фрагментов кода [2], особенно при распределенной памяти.

В настоящей работе исследованы проблемы создания распараллеливающих компиляторов на вычислительные системы с распределенной памятью. Автоматическое распараллеливание развито только для вычислительных систем с общей памятью (см., например, [3–6]).

Для вычислительных систем с распределенной памятью (ВСРП) самой длительной операцией является пересылка данных между процессорными элементами (ПЭ). Для высокопроизводительных кластеров такие пересылки могут даже вызвать замедление, поскольку их длительность в десятки раз больше времени выполнения вычислительных операций. Для эффективного отображения на ВСРП программа должна удовлетворять таким жестким требованиям, что ускорение может быть достигнуто для узкого класса программ, и разработка компилятора нецелесообразна. Но в последнее время появляются многоядерные микросхемы, называемые «суперкомпьютер на кристалле», с десятками, сотнями и даже тысячами ядер, большинство из которых ориентированы на реализацию нейронных сетей [7–15]. Пересылка данных между процессорными ядрами одной микросхемы требует значительно меньше времени, чем на коммуникационной сети кластера. Это означает расширение множества эффективно распараллеливаемых программ и делает целесообразным разработку распараллеливающих компиляторов.

Много систем автоматизации создания параллельного кода для ВСРП используют дописывание прагм в последовательную программу без автоматизации предварительного преобразования [16–18].

В [19] отмечена перспективность циклических пересылок данных для микросхем близкого будущего (с тысячами ядер). В [20] приведено много задач линейной алгебры и математической физики, для параллельного решения которых на ВСРП использованы циклические пересылки. Метод размещения данных с перекрытиями [2] существенно ускоряет параллельные итерационные алгоритмы, уменьшая количество пересылок при укрупнении множеств пересылаемых данных. В [22] показан дополнительный эффект ускорения от размещений с перекрытиями для микросхем «суперкомпьютер на кристалле». В [23] и [24] описаны блочно-аффинные размещения данных в распределенной памяти.

В [21] рассмотрена задача распараллеливания программного цикла на ВСРП, где для минимизации межпроцессорных пересылок по тексту программы построен вспомогательный граф «операторы-переменные» и приведены примеры, основанные на ОРС реализации [20].

Популярные оптимизирующие компиляторы (GCC, ICC, MS-Compiler, LLVM) распараллеливают программы только для вычислительных систем с общей памятью (ВСОП). Показано [25], что эти компиляторы имеют большой неиспользованный потенциал оптимизирующих преобразований. Эти результаты подтверждены в [26] и [27]. Можно полагать, что для микросхем supercomputer-on-a-chip этот потенциал неиспользуемых оптимизаций значительно больше.

Микросхемы «суперкомпьютер-на-кристалле» будут иметь большую производительность, чем многоядерные с общей памятью, и смогут выполнять новые классы программ с большим объемом вычислений. Отсутствие компиляторов на такие ВСРП сдерживает развитие новых высокопроизводительных систем.

В настоящей работе сформированы и обоснованы рекомендации к созданию оптимизирующих распараллеливающих компиляторов с последовательных программ языков высокого уровня для ВСРП, в частности, приведены оптимизирующие преобразования программ, которые должны использоваться в таком компиляторе. Рассмотрено также распараллеливание на ВСРП обхода дерева, которое задано рекурсивной функцией. Приведен пример распараллеливания гнезда циклов с условным оператором.

Подход, представленный ниже, отличается от других попыток автоматизировать отображение последовательных программ на ВСРП тем, что оптимальные

размещения массивов находятся среди блочно-аффинных размещений массивов [21], [23], которые, с одной стороны, могут быть описаны малым количеством параметров (пропорционально размерности массива), а, с другой стороны, покрывают размещения, широко используемые на практике. Расширение множества распараллеливаемых программ может быть достигнуто с помощью оптимизирующих преобразований программ, которые имеются в ОРС (Оптимизирующая распараллеливающая система) и в известных оптимизирующих компиляторах LLVM, GCC, ICC, MS-compiler.

1. МОДЕЛЬ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ С РАСПРЕДЕЛЕННОЙ ПАМЯТЬЮ

Будем рассматривать параллельную вычислительную систему с распределенной памятью (ВСРП), состоящую из процессоров и модулей памяти, которые соединяются коммуникационной системой, по которой можно пересылать данные. Пересылки данных по коммуникационной сети требуют много времени и составляют основную задачу оптимизации программ на ВСРП.

Существует множество конкретных вычислительных систем, которые удовлетворяют условиям описанной абстрактной модели. Многообразие таких вычислительных систем определяется разнообразием коммуникационных систем и вычислительных элементов и модулей памяти. ВСРП отличаются не только количеством модулей памяти и вычислительных элементов, но и их связями посредством коммуникационной сети, которые по длительности могут быть несимметричными. Распределенная память входит в состав вычислительных систем с программируемой архитектурой [28]. Как только будет разработан новый высокопроизводительный компьютер относительно небольших габаритов, сразу появится желание для некоторых важных задач объединить несколько таких компьютеров сетью – и появится новая ВСРП. Еще много архитектур будет придумано.

Многообразие архитектур ВСРП создает следующее требование к архитектуре преобразователей программ: должна быть система преобразований программ, которые полезны широкому множеству ВСРП, и должны быть библиотеки преобразований, ориентированные на конкретную целевую архитектуру. Эти требования аналогично тому, как устроены семейства компиляторов GCC и LLVM: есть библиотека оптимизирующих преобразований программ в промежуточном

представлении (middle end) и есть генераторы кода на целевую архитектуру (back end), которые тоже содержат оптимизирующие преобразования.

2. О ПРЕОБРАЗОВАНИЯХ ПРОГРАММ ДЛЯ ВСРП, КОТОРЫЕ НЕ НУЖНЫ ДЛЯ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ С ОБЩЕЙ ПАМЯТЬЮ

Отображение последовательных программ на ВСРП требует разработки дополнительных функций, которых нет в вычислительных системах с общей памятью (ВСОП), например: построение вспомогательного графа «операторы-переменные» для поиска минимального множества пересылок, генерация межпроцессорных пересылок, группировка пересылок, поиск оптимального размещение данных и кратного оптимального размещения данных в распределенной памяти с перекрытиями.

Оптимизирующие преобразования для ВСРП направлены, в первую очередь, на минимизацию пересылок между модулями системы. Можно выделить следующие функции и преобразования:

- Размещение данных в распределенной памяти.
- Поиск оптимального множества циклических пересылок с помощью ГОП (граф «операторы-переменные»).
- Транспонирование матриц, размещенных в распределенной памяти, и его обобщение на многомерные массивы.
- Размещение массивов данных кратными с перекрытиями.
- Группировка пересылок.
- Определение оптимального количества ПЭ.
- Оптимизация преобразований многомерных массивов: транспонирование и скашивание.

К преобразованиям, специфическим для ВСРП, можно отнести распознавание и вызов пересылок данных для многомерных массивов, таких как транспонирование и скашивание.

Транспонирование матриц используется и в библиотеках прикладных программ для обычных процессоров, поскольку от расположения элементов матрицы в оперативной памяти зависит скорость их считывания. Транспонирование может быть описано простой программой

```
for (int i = 0; i < n; i++) {
  for (int j = 0; j < n; j++) {
  Y(i,j) = X(j,i);
} }
```

В ВСРП транспонирование матрицы может быть полезно, если в одной части программы матрица должна быть размещена в распределенной памяти по строкам, а в другой – по столбцам. Если размерность матрицы равна количеству ПЭ и коммуникационная система поддерживает циклические пересылки данных, то транспонирование сводится к одновременным пересылкам диагональных элементов.

При скошенной форме хранения в ПЭ хранятся косые диагонали матрицы (диагонали, перпендикулярные к главной) [20]. Скошенная форма хранения позволяет обращаться одновременно как к элементам любой строки матрицы-клетки, так и к элементам любого столбца. Алгоритм приведения матрицы к скошенной форме может выглядеть следующим образом.

```
for (int i = 0; i < n; i++) {
  for (int j = 0; j < n; j++) {
  Y(i,j) = X((j+i) mod n ,i);
} }
```

3. ОПТИМИЗИРУЮЩИЕ ПРЕОБРАЗОВАНИЯ ПРОГРАММ, КОТОРЫЕ ПОЛЕЗНЫ ОДНОВРЕМЕННО И ДЛЯ ВСРП, И ДЛЯ ВСОП

В данном разделе рассмотрены предварительные преобразования последовательных программ перед отображением на вычислительную систему, которая может иметь как распределенную память, так и общую.

Оптимизировать следует участки кода, требующие больших объемов вычислений. Такими участками являются структуры вложенных циклов или рекурсивные функции. Обычные компиляторы LLVM, GCC, ICC, MS-compiler хорошо оптимизируют базовые блоки и самые глубоко вложенные циклы [30]. Такая оптимизация ориентирована на использование регистров, АЛУ и векторизацию. Но в последнее время в архитектуре компьютеров появилась сложная структура памяти даже в ВСОП (распределенная память ВСРП и так сегодня непосильна для

компиляторов). Для оптимизации использования сложной структуры памяти появилась потребность в преобразованиях циклов, которые могут содержать другие циклы (в первую очередь, тайлинг и скошенный тайлинг) [31], [30], а в преобразованиях данных – блочные размещения матриц в оперативной памяти [30], блочно-аффинные и размещения массивов и размещения с перекрытиями в распределенной памяти [30] и т. п.

К преобразованиям циклов следует отнести слияние, разбиение, гнездование, развертку, расщепление и др. Это преобразования, которые традиционно выполняются для самых глубоко вложенных циклов (innermost loop), но условия эффективного применения этих преобразований недостаточно разработаны к не самым глубоко вложенным циклам, и эти преобразования компиляторами не выполняются [32]. Есть преобразования, описанные в литературе, которые не выполняются в компиляторах, такие как ретайминг (сдвиги операторов между итерациями циклов) [30].

Диагональное расщепление двойного гнезда циклов позволяет избавиться от операции взятия остатка от деления при вычислениях с целыми числами. Оно встречается для быстрого вычисления сверток на циклической группе при восстановлении изображений [33], в теории кодирования и других приложениях. В компиляторах оно пока не реализуется. На микросхемах с распределенной локальной памятью названное преобразование актуально вместе с его применениями.

Не поддерживается компиляторами оптимизация обхода дерева вариантов, который возникает при рекурсивных вызовах функций, например, для многих задач, решаемых методом ветвей и границ. Нет инструментов, поддерживающих параллельный обход ветвей такого дерева [30].

Главная проблема современных оптимизирующих компиляторов на ВСОП – выбор оптимальной цепочки преобразований [30]. Для компиляторов на ВСРП эта проблема будет еще острее. Распараллеливание дерева обхода таких цепочек может быть полезно.

4. ПАРАЛЛЕЛЬНЫЙ ОБХОД ДЕРЕВА ПРИ РЕКУРСИВНОМ ВЫЗОВЕ ФУНКЦИИ

Рекурсивный вызов функции во многих случаях порождает процедуру обхода дерева. С другой стороны, обход дерева можно описывать рекурсивной функцией. Вызовы рекурсивных функций, как и программные циклы, во многих случаях требуют больших объемов вычислений и нуждаются в ускорениях. Обход дерева возникает в прикладных переборных задачах, многие из которых являются NP-трудными (например, известные задача коммивояжера или задача о ранце). В частности, обход дерева является составной частью метода ветвей и границ.

Многие переборные задачи не решаются точными алгоритмами, поскольку объемы вычислений не позволяют найти такое решение за приемлемые расходы времени и средств на современных компьютерах. Но приходит новое поколение вычислительных систем, и многие задачи, которые раньше не рассматривались как потенциально решаемые, теперь могут найти свое решение. Кроме того, на основе точных алгоритмов обхода дерева могут строиться более быстрые эвристические алгоритмы.

Естественный и обычно используемый метод распараллеливания обхода дерева – параллельное выполнение обхода разных ветвей [34], [35].

Рассмотрим оптимизацию обхода дерева рекурсивной функцией.

Будем рассматривать частичный инлайнинг рекурсивной функции. Вызов функции при трансляции в ассемблер создает дополнительные операции, такие как присваивания формальным параметрам их конкретных значений, после чего возможны упрощения арифметических выражений, и другие. Поэтому инлайнинг (подстановка вместо вызова функции тела этой функции) уменьшает количество операций и, как правило, ускоряет код. Если функция вызывается очень много раз, то выполнение инлайнинга для всех вызовов сильно увеличивает объем кода, что может приводить к замедлению программы.

У рекурсивной функции инлайнинг не исключает операцию вызова функции, и можно говорить только о частичном инлайнинге.

Рекурсивная функция создает обход дерева, если в ней есть по крайней мере два рекурсивных вызова, которые не лежат на альтернативных ветках условного оператора. Рассмотрим псевдокод такой функции, имеющей для простоты изложения два рекурсивных вызова.

```
Function F1
{B1;
Call F1;
B2;

Call F1;
B3;}
```

В этой функции можно сделать частичный инлайнинг обоих вызовов и получить новую функцию

```
Function F2
{B1;
      {B1;
      Call F1;
      B2;
      Call F1;
      B3;}
B2;
     {B1;
     Call F1;
     B2;
     Call F1;
     B3;}
B3;}
```

Таким образом, после частичного инлайнинга получено четыре вызова исходной функции F1. Можно повторить частичный инлайнинг несколько раз и получить много вызовов функции F1.

Выполнение исходной функции можно распараллелить, параллельно выполняя разные вызовы функции F1. Каждый вызов функции F1 соответствует обходу некоторой ветви дерева вычислений функции исходной программы.

Чем больше раз выполняется частичный инлайнинг, тем больше создается вызовов функций и, следовательно, параллельно выполняемых процессов.

Во многих алгоритмах, основанных на обходе дерева, особенно с использованием метода ветвей и границ, время выполнения разных ветвей дерева может очень сильно различаться. Иногда время обхода одной ветви дерева может быть больше времени обхода остальных вместе взятых. По этой причине параллельное выполнение разных ветвей дерева может оказаться плохо сбалансированным.

Если вызовов функций (ветвей дерева вычислений) больше, чем вычислительных устройств (процессоров, процессорных элементов, процессорных ядер, …), то некоторым устройствам придется выполнять по нескольку вызовов. Но заранее не известно, какой вызов функции F1 как долго будет обрабатываться.

Для улучшения балансировки нагрузки вычислительных устройств удобно использовать технологии параллельного программирования типа TPL или Intl TBB, в которых создается очередь заданий (вызовов функций), и вычислительные устройства получают новые задания из очереди, как только они освобождаются от выполнения предыдущего.

Чем большее количество мелких заданий в очереди, тем лучше может быть сбалансирована нагрузка вычислительных устройств. Но, с другой стороны, увеличиваются накладные расходы с запуском на выполнение очередного задания из очереди. Запуск задания может быть связан не только с командами запуска, но и с подготовкой данных, особенно при распределенной памяти.

Будем минимизировать время на пересылки заданий между ПЭ. Будет полагать, что ПЭ соединяются шиной или кольцевой сетью. Пусть в некоторый момент работы алгоритма каждый ПЭ имеет некоторое множество заданий и освободился один ПЭ (количество его заданий равно нулю), которому требуется переслать задание. Конечно, быстрее всего переслать задание из соседнего ПЭ. Но шина (или кольцевая шина) позволяет в одно и то же время делать несколько пересылок в одном направлении. Этим самым можно подготовиться к минимизации времени следующих пересылок.

Наилучшим представляется распределение количества заданий, при котором в каждом ПЭ одинаковое количество заданий. Таким образом, будем делать

пересылки, при которых минимизируется среднее квадратичное отклонение от среднего количества заданий в ПЭ.

Пример

Номер ПЭ	0	1	2	3	4	5	6	7
К-во заданий	4	4	2	3	0	2	4	1

Среднее количество заданий в ПЭ равно 2.5.

Разумной (оптимальной) представляется следующая пересылка заданий слева направо:

ПЭ0→ПЭ1; ПЭ1→ПЭ2; ПЭ3→ПЭ4; ПЭ6→ПЭ7.

После такой пересылки распределение заданий будет таким

Номер ПЭ	0	1	2	3	4	5	6	7
К-во заданий	3	3	2	2	1	2	3	2

Среднее квадратичное отклонение от среднего количества заданий равно

$$(1.5^2+1.5^2+0.5^2+0.5^2+1.5^2+0.5^2+0.5^2+0.5^2)/8=1.0.$$

Приведем алгоритм для поиска оптимальной пересылки заданий по шине.

Задача алгоритма – не только загрузить ПЭ, у которого нет заданий, но и в это же время (этой же пересылкой) уменьшить среднее квадратичное отклонение от среднего количества заданий. Это приведет к уменьшению времени выполнения при предположении, что время выполнения всех заданий одинаково.

Будем отдельно искать оптимальный алгоритм пересылки данных по шине слева направо и справа налево, затем выберем из них лучший. Ясно, что достаточно описать только один алгоритм, например, поиска оптимальной пересылки слева направо.

Пусть количество ПЭ равно n, и они занумерованы от 0 до (n-1). Обозначим b_j количество заданий в ПЭ с номером j. Ясно, что b_j>=0. Обозначим через x_j количество заданий, пересылаемых из ПЭ с номером j в ПЭ с номером (j+1). Величина x_j принимает значения 0 или 1. В результате работы алгоритма определяются все значения x_j.

Алгоритм (жадный)

Если у каждого ПЭ количество заданий не больше 1, то конец работы алгоритма (т. е. ПЭ, у которых есть задания, завершают их выполнение, пересылки заданий нецелесообразны).

Полагаем x_{n-1}=0.

Просматриваем все ПЭ с номерами j справа налево, начиная с (n-2)-го до 0-го. Если $b_j > b_{j+1} - x_{j+1}$, то x_j=1, иначе x_j=0.

В результате действия данного алгоритма не увеличивается количество заданий каждого ПЭ. Если хотя бы у одного ПЭ количество заданий больше 1 при том, что у какого то ПЭ заданий 0, то данный алгоритм найдет пересылку либо слева направо, либо справа налево, которая уменьшит среднее квадратичное отклонение количества заданий от среднего количества заданий.

5. О СОЗДАНИИ АРХИТЕКТУРЫ РАСПАРАЛЛЕЛИВАЮЩЕЙ СИСТЕМЫ НА ВСРП

Даже для ВСОП распараллеливающий компилятор – это программный продукт, большой по объему кода, обладающий сложной логикой, требующий очень высокой квалификации разработчиков и значительного времени разработки. По этой причине при разработке компилятора на новый процессор используются большие части кода компиляторов, уже созданных для прежних вычислительных систем (например, оптимизирующие преобразования LLVM). Оригинальными являются генератор кода и некоторые комбинации существующих преобразований. Такой подход по экономическим причинам неизбежен (с дополнительными усложнениями) и для компиляторов на ВСРП.

Специфические преобразования программ на ВСРП предполагают выделение их в отдельную библиотеку. Поскольку для многих архитектур ВСРП выход компилятора должен содержать участки кода, выполняемые на ВСОП, результатом таких преобразований должен быть, скорее всего, высокоуровневый язык, к которому можно применить существующий компилятор на ВСОП (например, ICC или C-to-CUDA). Поскольку самый быстрый код создается программами языков C и ФОРТРАН, то на эти языки и должна быть рассчитана распараллеливающая система на ВСРП. Такая система будет совместима со всеми компиляторами. Даже для ВСОП преобразования гнезд циклов удобнее выполнять с выходным высокоуровневым языком [36].

Многообразие архитектур ВСРП предполагает создание, в первую очередь, преобразований программ, полезных для многих ВСРП: размещение данных в

распределенной памяти (блочно-аффинные), алгоритмы поиска минимального множества циклических пересылок и рассылок данных (broadcast).

Внутреннее (промежуточное) представление обсуждаемой распараллеливающей системы желательно делать высокоуровневым. Высокоуровневое внутреннее представление позволит создавать выходной код, более близкий (узнаваемый) к исходному коду разработчика последовательной программы. К такому коду легче применять диалоговый режим уточнения зависимостей [29]. Известные распараллеливающие системы, выдающие код на высокоуровневом языке, имеют, как правило, высокоуровневое внутреннее представление: ROSE, SUIF, OPS, PLUTO.

6. ОПТИМИЗИРУЮЩИЕ КОМПИЛЯТОРЫ VS БИБЛИОТЕК

Разработчики процессоров ВСОП в качестве системного ПО выпускают компиляторы и библиотеки. Для создания высокопроизводительных приложений следует использовать библиотеки. Но они разрабатываются не для всех приложений. При выпуске новой микросхемы необходимо все библиотеки переписывать.

Для ВСРП создание библиотек требует существенно больших усилий, чем для ВСОП, поскольку исходные данные для библиотечных функций в разных случаях могут быть размещены по-разному. Рассмотрим, например, задачу перемножения матриц $C=A*B$. Если размерность исходных матриц равна количеству ПЭ, то в случаях, рассматриваемых на практике, каждая матрица может размещаться либо «по строкам», либо «по столбцам», либо «в скошенной форме». В этом случае нужно 9 библиотечных функций. Но, если размерности матриц больше, чем количество ПЭ, то матрицы могут размещаться полосами строк или столбцов, что увеличивает количество библиотечных функций.

Если ВСРП используется как ускоритель некоторого класса приложений, то библиотеки могут иметь преимущество по сравнению с компилятором. Если ВСРП предполагается использовать для переноса программ с других вычислительных систем или для разработки новых приложений, то использование компилятора более предпочтительно.

Есть еще одна функция для использования высокоуровневого преобразователя программ – проектирование новых вычислительных систем. Действительно,

эти системы при проектировании подстраиваются под некоторые бенчмарки. Бенчмарки – это конкретные программные реализации некоторых полезных алгоритмов. Но алгоритмы могут быть реализованы не единственным образом. Высокоуровневый преобразователь программ может представить семейство программ, эквивалентных исходной (бенчмарку). Проектируемая вычислительная система может ориентироваться на любой экземпляр семейства.

7. ЗАКЛЮЧЕНИЕ

В настоящей статье приведены основные элементы проекта системы преобразования программ для создания распараллеливающих компиляторов на вычислительные системы с распределенной памятью.

Существует много приложений, для которых не видно предела в повышении производительности вычислительных систем: прогноз погоды и изменений климата, отслеживание угроз столкновения метеоритов с Землей, планирование экономики, создание новых лекарств, решение задач метагеномики и др. Потребность в новых высокопроизводительных программно-аппаратных комплексах будет у общества еще многие годы. Дорогой проект создания высокоуровневого преобразователя программ для ВСРП является вполне экономически целесообразным.

Благодарности

Исследование выполнено за счет гранта Российского научного фонда № 22-21-00671, https://rscf.ru/project/22-21-00671/.

СПИСОК ЛИТЕРАТУРЫ

1. *Bondhugula U.* Automatic distributed-memory parallelization and codegeneration using the polyhedral framework, Technical report, ISc-CSA-TR-2011-3, 2011, 10 p.

2. *Ammaev S.G., Gervich L.R., Steinberg B.Y.* Combining parallelization with overlaps and optimization of cache memory usage. PaCT 2017: Parallel Computing Technologies, Lecture Notes in Computer Science. Vol. 10421. P. 257–264.

3. Векторизация программ // Векторизация программ: теория, методы, реализация / Сборник переводов статей. М.: Мир, 1991. С. 246–267.

4. *Moldovanova O.V., Kurnosov M.G.* Auto-Vectorization of Loops on Intel 64 and Intel Xeon Phi: Analysis and Evaluation International Conference on Parallel Computing Technologies. PaCT 2017: Parallel Computing Technologies, Lecture Notes in Computer Science. Vol. 10421. P. 143–150.

5. Nvidia compilers. URL: https://developer.nvidia.com/hpc-compiler

6. *Peng Di, Ding Ye, Yu Su, Yulei Sui, Jingling Xue.* Automatic Parallelization of Tiled Loop Nests with Enhanced Fine-Grained Parallelism on GPUs. 2012. 41st International Conference on Parallel Computing.

7. SoC Esperanto. URL: https://www.esperanto.ai/technology/

8. Процессор НТЦ «Модуль». URL: https://www.cnews.ru/news/top/2019-03-06_svet_uvidel_moshchnejshij_ rossijskij_nejroprotsessor (дата обр. 26.03.2022).

9. *Peckham O. SambaNova.* Launches Second-Gen DataScale System. URL: https://www.hpcwire.com/2022/09/14/sambanova-launches-second-gen-datascalesystem/.

10. *Елизаров Г.С., Конотопцев В.Н., Корнеев В.В.* Специализированные большие интегральные схемы для реализации нейросетевого вывода. XXII международная конференция «Харитоновские тематические научные чтения». Суперкомпьютерное моделирование и искусственный интеллект: труды / Редактор Р.М. Шагалиев. Саров: ФГУП «РФЯЦ-ВНИЭФ», 2022. С. 181–184.

11. *Корнеев В.В.* Направления повышения производительности нейросетевых вычислений // Программная инженерия. 2020. Т. 11, № 1. С. 21–25.

12. *Yen I.E., Xiao Zh., Xu D.* S4: a High-sparsity, High-performance AI Accelerator // arXiv:2207.08006v1 [cs.AR] 16 Jul 2022.

13. *Gale T., Elsen E., Hooker S.* The state of sparsity in deep neural networks // arXiv preprint arXiv:1902.09574, 2019.

14. Intelligence Processing Unit. URL: https://www.graphcore.ai/products/ipu.

15. *Jia Zh., Tillman B., Maggioni M., Scarpazza D.P.* Dissecting the Graphcore IPU Architecture via Microbenchmarking // Technical Report. December 7, 2019. arXiv:1912.03413v1 [cs.DC] 7 Dec 2019. 91 p.

16. DVM-система разработки параллельных программ. URL: http://dvm-system.org/ru/about/

17. *Kataev N., Kolganov A.* Additional Parallelization of Existing MPI Programs Using SAPFOR. In: Malyshkin V. (Ed.) Parallel Computing Technologies. PaCT 2021. Lecture Notes in Computer Science. 2021. Vol. 12942. Springer, Cham.
URL: https://doi.org/10.1007/978-3-030-86359-3_4

18. *Kwon D., Han S., Kim H.* MPI backend for an automatic parallelizing compiler // Proceedings Fourth International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN'99). 06.1999. P. 152–157.

19. *Корнеев В.В.* Параллельное программирование // Программная инженерия. 2022. Т. 13, № 1. С. 3–16.

20. *Прангишвили И.В., Виленкин С.Я., Медведев И.Л.* Параллельные вычислительные системы с общим управлением. М.: Энергоатомиздат, 1983. 312 с.

21. *Krivosheev N.M., Steinberg B.Ya.* Algorithm for searching minimum inter-node data transfers // Procedia Computer Science, 10th International Young Scientist Conference on Computational Science, YSC 2021, 1–3 July 2021. P. 306–313.

22. *Gervich L.R., Steinberg B.Ya.* Automation of the Application of Data Distribution with Overlapping in Distributed Memory // Bulletin of the South Ural State University. Ser. Mathematical Modeling, Programming & Computer Software (Bulletin SUSU MMCS). 2023. Vol. 16, no. 1. P. 59–68.

23. *Штейнберг Б.Я.* Блочно-аффинные размещения данных в параллельной памяти // Информационные технологии. 2010. №6. С. 36–41.

24. *Штейнберг Б.Я.* Оптимизация размещения данных в параллельной памяти. Ростов-на-Дону, Изд-во Южного федерального университета, 2010. 255 с.

25. *Gong Z., Chen Z., Szaday Z., Wong D., Sura Z., Watkinson N., Maleki S., Padua D., Veidenbaum A., Nicolau A.* An empirical study of the effect of source-level loop transformations on compiler stability // Proceedings of the ACM on Programming Languages. 11.2018. P. 1–29.

26. *Steinberg B.Ya., Steinberg O.B., Oganesyan P.A., Vasilenko A.A., Veselovskiy V.V., Zhivykh N.A.* Fast Solvers for Systems of Linear Equations with Block-Band Matrices // East Asian Journal on Applied Mathematics. 2023. Vol. 13, No. 1. P. 47–58.

27. *Vasilenko A., Veselovskiy V., Metelitsa E., Zhivykh N., Steinberg B., Steinberg O.* Precompiler for the ACELAN-COMPOS Package Solvers // In: Malyshkin V. (Ed.)

Parallel Computing Technologies. PaCT 2021. Lecture Notes in Computer Science. 2021. Vol. 12942. P. 103–116. Springer, Cham.
URL: https://doi.org/10.1007/978-3-030-86359-3_8

28. *Dordopulo A.I., Levin I.I., Gudkov V.A., Gulenok A.A.* High-Level Synthesis of Scalable Solutions from C-Programs for Reconfigurable Computer Systems // In: Malyshkin V. (Ed.) Parallel Computing Technologies. PaCT 2021. Lecture Notes in Computer Science. 2021. Vol. 12942. Springer, Cham.
URL: https://doi.org/10.1007/978-3-030-86359-3_7

29. *Штейнберг Б.Я.* Блочно рекуррентное размещение матрицы для параллельного выполнения алгоритма Флойда // Известия ВУЗов. Северокавказский регион. Естественные науки. 2010. №5. С. 31–33.

30. *Штейнберг Б.Я., Штейнберг О.Б.* Преобразования программ – фундаментальная основа создания оптимизирующих распараллеливающих компиляторов // Программные системы: теория и приложения. 2021. Т. 12, № 1(48). С. 21–113. URL: http://psta.psiras.ru/read/psta2021_1_21-113.pdf

31. *Wolfe M.* More Iteration Space Tiling // Supercomputing. Reno, 1989. P. 655–664.

32. *Штейнберг Б.Я., Штейнберг О.Б., Василенко А.А.* Слияние циклов для локализации данных // Программные системы. Теория и приложения. 2020. Т. 11, №3. С. 17–31. URL: https://doi.org/10.25209/2079-3316-2020-11-3-17-31

33. *Козак А.В., Штейнберг Б.Я., Штейнберг О.Б.* Алгоритм восстановления смазанного изображения, полученного вращающейся под углом к горизонту камерой // Компьютерная оптика. 2020. Т. 44, № 2. С. 229–235.

34. *Burkhovetskiy V.V., Steinberg B.Ya.* Parallelizing an Exact Algorithm for the Traveling Salesman Problem // Procedia Computer Science, 6-th International Young Scientist Conference on Computational Science, YSC 2017, Procedia Computer Science. 2017. Vol. 119. P. 97–102.
URL: http://authors.elsevier.com/sd/article/S187705091732375X

35. *Бурховецкий В.В., Штейнберг Б.Я.* Стратегия использования крупных заданий при параллельном обходе дерева // Языки программирования и компиля-

торы. Труды Всероссийской научной конференции памяти А.Л. Фуксмана. 3–5 апреля 2017, Южный федеральный университет, г. Ростов-на-Дону: Изд-во Южного федерального университета. 2017. С. 66–70.

36. *Zhiyuan Li, Yonghong Song.* Automatic Tiling of Iterative Stencil Loops // ACM Transactions on Programming Languages and Systems. 2004. Vol. 26, No. 6. P. 975–1028.

37. *Gervich L.R., Guda S.A., Dubrov D.V., Ibragimov R.A., Metelitsa E.A., Mikhailuts Y.M., Paterikin A.E., Petrenko V.V., Skapenko I.R., Steinberg B.Ya., Steinberg O.B., Yakovlev V.A., Yurushkin M.V.* How OPS (Optimizing Parallelizing System) May be Useful for Clang // CEE-SECR ’2017, October 20–21, 2017, St.-Peterburg, Russian Federation. Proceedings of the 13th Central & Eastern European Software Engineering Conference in Russia ACM New York, NY, USA. 2017.
URL: https://dl.acm.org/citation.cfm?id=3166094&picked=prox

ON THE WAY TO CREATING PARALLELIZING COMPILERS FOR COMPUTING SYSTEMS WITH DISTRIBUTED MEMORY

B. Ya. Steinberg[0000-0001-8146-0479]

Southern federal university

byshtyaynberg@sfedu.ru

Abstract

The conditions for creating optimizing parallelizing compilers for computing systems with distributed memory are described. Target computing systems are microcircuits of the “supercomputer on a chip” type. Both optimizing program transformations specific to systems with distributed memory and those transformations that are needed both for computing systems with distributed memory and for computing systems with shared memory are presented. The issues of minimizing interprocessor transfers when parallelizing a recursive function are discussed. The main approach to creating such compilers is block-affine data placement in distributed memory with minimization of inter-processor transfers. It is shown that parallelizing compilers for computing systems with distributed memory should be created on the basis of a high-level internal representation and a high-level output language.

***Keywords**: automatic parallelization, distributed memory, program transformation, data distribution, data interchange*

REFERENCES

1. *Bondhugula U.* Automatic distributed-memory parallelization and codegeneration using the polyhedral framework, Technical report, ISc-CSA-TR-2011-3, 2011, 10 p.

2. *Ammaev S.G., Gervich L.R., Steinberg B.Y.* Combining parallelization with overlaps and optimization of cache memory usage. PaCT 2017: Parallel Computing Technologies, Lecture Notes in Computer Science. Vol. 10421. P. 257–264.

3. *Vektorizaciya programm* // Vektorizaciya programm: teoriya, metody, realizaciya. / Sbornik perevodov statej. M.: Mir, 1991. S. 246–267.

4. *Moldovanova O.V., Kurnosov M.G.* Auto-Vectorization of Loops on Intel 64 and Intel Xeon Phi: Analysis and Evaluation International Conference on Parallel Computing Technologies. PaCT 2017: Parallel Computing Technologies, Lecture Notes in Computer Science. Vol. 10421. P. 143–150.

5. Nvidia compilers. URL: https://developer.nvidia.com/hpc-compiler

6. *Peng Di, Ding Ye, Yu Su, Yulei Sui, Jingling Xue.* Automatic Parallelization of Tiled Loop Nests with Enhanced Fine-Grained Parallelism on GPUs. 2012. 41st International Conference on Parallel Computing.

7. SoC Esperanto. URL: https://www.esperanto.ai/technology/

8. Processor NTC “Modul”.
URL: https://www.cnews.ru/news/top/2019-03-06_svet_uvidel_moshchnejshij_rossijskij_nejroprotsessor

9. *Peckham O. SambaNova.* Launches Second-Gen DataScale System.
URL: https://www.hpcwire.com/2022/09/14/sambanova-launches-second-gen-datascalesystem/.

10. *Elizarov G.S., Konotoptsev V.N., Korneev V.V.* Specialized large integrated circuits for the implementation of neural network inference // XXII International conference "Kharitonov thematic scientific readings "Supercomputer modeling and Artificial-Intelligence": proceedings / Edited by R.M. Shagaliev. Sarov: FSUE "RFSC-VNIIEF", 2022. P. 181–184.

11. *Korneev V.V.* Approaches to improving the performance of neural network computing // Programmnaya ingeneria. 2020. Vol. 11. No. 1. P. 21–25.
URL: https://doi.org/10.17587/prin.11.21-25.

12. *Yen I.E., Xiao Zh., Xu D.* S4: a High-sparsity, High-performance AI Accelerator // arXiv:2207.08006v1 [cs.AR] 16 Jul 2022.

13. *Gale T., Elsen E., Hooker S.* The state of sparsity in deep neural networks // arXiv preprint arXiv:1902.09574, 2019.

14. Intelligence Processing Unit.
URL: https://www.graphcore.ai/products/ipu.

15. *Jia Zh., Tillman B., Maggioni M., Scarpazza D.P.* Dissecting the Graphcore IPU Architecture via Microbenchmarking // Technical Report. December 7, 2019. arXiv:1912.03413v1 [cs.DC] 7 Dec 2019. 91 p.

16. DVM-sistema razrabotki parallel'nyh programm.
URL: http://dvm-system.org/ru/about/

17. *Kataev N., Kolganov A.* Additional Parallelization of Existing MPI Programs Using SAPFOR. In: Malyshkin V. (Ed.) Parallel Computing Technologies. PaCT 2021. Lecture Notes in Computer Science. 2021. Vol. 12942. Springer, Cham.
URL: https://doi.org/10.1007/978-3-030-86359-3_4

18. *Kwon D., Han S., Kim H.* MPI backend for an automatic parallelizing compiler // Proceedings Fourth International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN'99). 06.1999. P. 152–157.

19. *Korneev V.V.* Parallel'noe programmirovanie // Programmnaya inzhenerіya. 2022. T. 13, № 1. S. 3–16.

20. *Prangishvili I.V., Vilenkin S.YA., Medvedev I.L.* Parallel'nye vychislitel'nye sistemy s obshchim upravleniem. M.: Energoatomizdat, 1983. 312 p.

21. *Krivosheev N.M., Steinberg B.Ya.* Algorithm for searching minimum internode data transfers // Procedia Computer Science, 10th International Young Scientist Conference on Computational Science, YSC 2021, 1–3 July 2021. P. 306–313.

22. *Gervich L.R., Steinberg B.Ya.* Automation of the Application of Data Distribution with Overlapping in Distributed Memory // Bulletin of the South Ural State University. Ser. Mathematical Modelling, Programming & Computer Software (Bulletin SUSU MMCS). 2023. Vol. 16, no. 1. P. 59–68.

23. *Shtejnberg B.Ya.* Blochno-affinnye razmeshcheniya dannyh v parallel'noj pamyati // Informacionnye tekhnologii. 2010. No 6. S. 36–41.

24. *Shtejnberg B.Ya.* Optimizaciya razmeshcheniya dannyh v parallel'noj pamyati. Rostov-na-Donu: Izd-vo Yuzhnogo federal'nogo universiteta, 2010. 255 s.

25. *Gong Z., Chen Z., Szaday Z., Wong D., Sura Z., Watkinson N., Maleki S., Padua D., Veidenbaum A., Nicolau A.* An empirical study of the effect of source-level loop transformations on compiler stability // Proceedings of the ACM on Programming Languages. 11.2018. P. 1–29.

26. *Steinberg B.Ya., Steinberg O.B., Oganesyan P.A., Vasilenko A.A., Veselovskiy V.V., Zhivykh N.A.* Fast Solvers for Systems of Linear Equations with Block-Band Matrices // East Asian Journal on Applied Mathematics. 2023. Vol. 13, No. 1. P. 47–58.

27. *Vasilenko A., Veselovskiy V., Metelitsa E., Zhivykh N., Steinberg B., Steinberg O.* Precompiler for the ACELAN-COMPOS Package Solvers // In: Malyshkin V. (Ed.) Parallel Computing Technologies. PaCT 2021. Lecture Notes in Computer Science. 2021. Vol. 12942. P. 103–116. Springer, Cham.
URL: https://doi.org/10.1007/978-3-030-86359-3_8

28. *Dordopulo A.I., Levin I.I., Gudkov V.A., Gulenok A.A.* High-Level Synthesis of Scalable Solutions from C-Programs for Reconfigurable Computer Systems // In: Malyshkin V. (Ed.) Parallel Computing Technologies. PaCT 2021. Lecture Notes in Computer Science. 2021. Vol. 12942. Springer, Cham. URL: https://doi.org/10.1007/978-3-030-86359-3_7*29.*

29. *Gervich L.R., Guda S.A., Dubrov D.V., Ibragimov R.A., Metelitsa E.A., Mikhailuts Y.M., Paterikin A.E., Petrenko V.V., Skapenko I.R., Steinberg B.Ya., Steinberg O.B., Yakovlev V.A., Yurushkin M.V.* How OPS (Optimizing Parallelizing System) May be Useful for Clang // CEE-SECR ’2017, October 20–21, 2017, St.-Peterburg, Russian Federation. Proceedings of the 13th Central & Eastern European Software Engineering Conference in Russia ACM New York, NY, USA. 2017.
URL: https://dl.acm.org/citation.cfm?id=3166094&picked=prox

30. *Shtejnberg B.Ya., Shtejnberg O.B.* Preobrazovaniya programm – fundamental'naya osnova sozdaniya optimiziruyushchih rasparallelivayushchih kompilyatorov // Programmnye sistemy: teoriya i prilozheniya. 2021. T. 12, № 1(48). S. 21–113.
URL: http://psta.psiras.ru/read/psta2021_1_21-113.pdf

31. *Wolfe M.* More Iteration Space Tiling // Supercomputing. Reno, 1989. P. 655–664.

32. *Shtejnberg B.Ya., Shtejnberg O.B., Vasilenko A.A.* Sliyanie ciklov dlya lokalizacii dannyh // Programmnye sistemy. Teoriya i prilozheniya. 2020. T. 11, №3. S. 17–31. URL: https://doi.org/10.25209/2079-3316-2020-11-3-17-31

33. *Kozak A.V., Shtejnberg B.Ya., Shtejnberg O.B.* Algoritm vosstanovleniya sma-zannogo izobrazheniya, poluchennogo vrashchayushchejsya pod uglom k gorizontu kameroj // Komp'yuternaya optika. 2020. T. 44, № 2. S. 229–235.
URL: https://doi.org/10.18287/2412–6179-CO-598.
URL: http://www.computeroptics.smr.ru/

34. *Burkhovetskiy V.V., Steinberg B.Ya.* Parallelizing an Exact Algorithm for the Traveling Salesman Problem // Procedia Computer Science, 6-th International Young Scientist Conference on Computational Science, YSC 2017, Procedia Computer Science. 2017. Vol. 119. P. 97-102.
URL: http://authors.elsevier.com/sd/article/S187705091732375X

35. *Burhoveckij V.V., Shteinberg B.Ya.* Strategiya ispol'zovaniya krupnyh zadanij pri parallel'nom obhode dereva. Yazyki programmirovaniya i kompilyatory. Trudy Vserossijskoj nauchnoj konferencii pamyati A.L. Fuksmana. 3–5 aprelya 2017, Yuzhnyj federal'nyj universitet, g. Rostov-na-Donu: Izdatel'stvo Yuzhnogo federal'nogo universiteta, 2017. S. 66–70.

36. *Zhiyuan Li, Yonghong Song.* Automatic Tiling of Iterative Stencil Loops // ACM Transactions on Programming Languages and Systems. 2004. Vol. 26, No. 6. P. 975–1028.

37. *Gervich L.R., Guda S.A., Dubrov D.V., Ibragimov R.A., Metelitsa E.A., Mikhailuts Y.M., Paterikin A.E., Petrenko V.V., Skapenko I.R., Steinberg B.Ya., Steinberg O.B., Yakovlev V.A., Yurushkin M.V.* How OPS (Optimizing Parallelizing System) May be Useful for Clang // CEE-SECR ’2017, October 20–21, 2017, St.-Peterburg, Russian Federation. Proceedings of the 13th Central & Eastern European Software Engineering Conference in Russia ACM New York, NY, USA. 2017.
URL: https://dl.acm.org/citation.cfm?id=3166094&picked=prox

СВЕДЕНИЯ ОБ АВТОРЕ

Штейнберг Борис Яковлевич – д. т. н, зав. каф., с. н. с. Института математики, механики и компьютерных наук Южного федерального университета.

Boris Yakovlevich STEINBERG – Doctor of Computer Science, head of chair in department of Mathematics, mechanics and computer science of Southern federal university.

e-mail: borsteinb@mail.ru
ORCID: 0000-0001-8146-0479

Материал поступил в редакцию 23 декабря 2023 года