ДИАЛОГОВЫЙ ВЫСОКОУРОВНЕВЫЙ ОПТИМИЗИРУЮЩИЙ РАСПАРАЛЛЕЛИВАТЕЛЬ (ДВОР)

Б.Я. Штейнберг, А.А. Абрамов, Е.В. Алымова, А.П. Баглий, С.А. Гуда, Д.В. Дубров, Е.Н. Кравченко, Р.И. Морылев, З.Я. Нис, В.В. Петренко, С.В. Полуян, И.С. Скиба, В.Н. Шаповалов, О.Б. Штейнберг, Р.Б. Штейнберг, М.В. Юрушкин

Введение.

Проект ДВОР [1] представляет собой программный инструментальный комплекс, ориентированный на разработку распараллеливающих и оптимизирующих компиляторов с языков программирования высокого уровня (С, Фортран и др.), систем полуавтоматического распараллеливания. В ДВОР включены как парсеры собственной разработки, так и сторонние [2], обеспечивающие построение внутреннего представления программ из языков высокого уровня. ДВОР содержит большую библиотеку оптимизирующих преобразований, использующих граф информационных связей и решётчатый граф [3], и ориентированных на различные целевые параллельные архитектуры.

Высокоуровневое внутреннее представление.

Внутреннее представление программ в ДВОР ориентировано на разработку высокоуровневых распараллеливающих преобразований процедурных языков программирования. К проекту ДВОР предъявлены требования многоязыковости (Си и Фортран), общей библиотеки преобразований для различных языков, возможности диалоговой компиляции. Эти требования иногда противоречат друг другу. Возникающие противоречия разрешаются структурой внутреннего представления. Внутреннее представление является многоуровневым, с разной детализацией программы на каждом уровне. Верхний уровень для каждого входного языка индивидуальный и удобен для разбора. Второй уровень общий для всех входных языков ориентирован на преобразования программ. В языках Си и Фортран по-разному определены массивы. Второй уровень представления сглаживает эти различия, чтобы можно было применить общий алгоритм для определения зависимостей. На этом уровне разобранная программа хранится в виде четырех деревьев: дерево идентификаторов, дерево типов, дерево операторов и дерево выражений. Такой высокий уровень удобен для диалоговой формы преобразований.

Некоторые преобразования и их особенности.

Система ДВОР предназначена для проведения в первую очередь высокоуровневых преобразований. Обеспечивая ряд преимуществ, высокий уровень создает проблемы нарушения ограничений статической семантики (при формальном применении преобразований).

Пример. Преобразование "Гнездование циклов" заменяет цикл:

```
for (i = 0 ; i < n; i = i + 1)
  loopBody(i);
на следующий фрагмент программы:
for (j = 0; j < h; j = j + 1)
  for (i = 0; i < n / h; i = i + 1)
    loopBody(i + j * (n / h));
for (i = 0; i < n - (n / h) * h; i = i + 1)
  loopBody((n / h) * h + i);
```

Данное преобразование используется при распараллеливании цикла на суперкомпьютере типа MIMD с количеством процессоров h. Если loopBody(i) содержит помеченные операторы, то в копиях тела цикла должны вводиться новые метки и изменяться ссылки на метки. Ситуация с метками для гнездования цикла аналогична и для других преобразований, в которых копируется код фрагмента: развертка цикла, раскрутка цикла, расщепление цикла, вынос условного оператора из цикла. Подобные примеры возможных ошибок приводятся в [5]. Для ДВОР разрабатывается система, которая позволяет обнаруживать все возможные подобные нарушения. Система основана на формализации правил-ограничений статической семантики и алгоритма преобразования и не является статистической, как системы тестирования.

Преобразование "Разрезание цикла" (loop distribution) является обратным к преобразованию слияние циклов (loop fusion). Это преобразование может быть полезно, например, при отображении программы на вычислительную систему с конвейерной архитектурой. Применению данного преобразования могут мешать зависимости по данным, ведущие «снизу-вверх», которые иногда можно устранить при помощи преобразований «введение временных массивов» и «растягивание скаляров» [4]. В алгоритме разрезания цикла, используемом в системе ДВОР, строится граф информационных связей, по которому строится фактор-граф. В полученном фактор-графе строится правильная нумерация. Преобразование разрезание цикла может иметь разные

вариации. Например, разрезание в указанном месте или на максимальное количество частей. Кроме того, при желании можно не использовать вспомогательные преобразования. В этом случае, например, не возникнет расходов памяти в связи с появлением нового массива.

Преобразование «Разрезание условного оператора» (if distribution) заменяет один условный оператор двумя, при этом блок ветки then исходного условного оператора разбивается на две части и каждая из частей становится блоком соответствующего результирующего условного оператора. В ряде ситуаций для хранения значения условия исходного условного оператора вводится дополнительная переменная. Это происходит в случаях: 1) наличия вызовов функций в исходном фрагменте; 2) наличия операторов goto, break, continue; 3) существования дуг выходной и анти- зависимостей, ведущих из условия условного оператора в 1-ю часть блока ветки then исходного условного оператора.

Всякое преобразование программы состоит из формального преобразования текста и различных проверок:

- Распознавание фрагмента, к которому применяется преобразование.
- Проверка сохранения синтаксиса и семантики
- Контроль эквивалентности преобразования данных
- Проверка целесообразности
- Выбор лучшей оптимизации выбранного фрагмента

Формальные преобразования имеют вид: «Вставить», «Удалить», «Копировать», «Заменить». Эти изменения относятся и к операторам, и к выражениям, и к фрагментам кода (блокам, функциям). Используются операции «Создать» новые имена переменных, меток, функций. Результирующий фрагмент программы должен быть в скобках (операторных или для операций соответственно).

Проверки могут выполняться автоматически и в диалоге. Для контроля эквивалентности преобразований кроме традиционного графа информационных связей используется и решетчатый граф программы.

Разработано расширение скалярной SSA-формы (static single assignment form, форма со статически однократным присваиванием) [6] на код с массивами. SSA-форма для массивов строится с помощью нечёткого анализа потока данных (Fuzzy Array Dataflow Analysis) [7]. В SSA-форме для массивов некоторые скалярные преобразования расширяют свою область применения: удаление мертвого кода, протягивание констант, использование общих подвыражений, распределение регистров и др. Получено новое оптимизирующее преобразование - удаление полумертвого кода (оператор в теле цикла полумертвый, если на некоторых итерациях можно его не выполнять). Скалярная SSA-форма не позволяет выполнить такое преобразование.

В системе ДВОР для гнезд циклов с внешними переменными в индексных выражениях и в границах счетчиков циклов строится решетчатый граф (см. [3, 8]). Это позволяет распараллеливать не только ParDo циклы, но и параллельно выполнять точки пространства итераций, а также и такие преобразования ДВОР, как расщепление многомерных гнезд циклов, экспансия массивов (см. [9]), подстановка массивов, которые не могут выполняться на основе традиционного графа зависимостей по данным.

Разработан автоматический генератор тестов [10] для проверки корректности преобразований. На вход генератор получает формальное описание грамматики целевого языка (C или Fortran) и файл конфигурации для конкретного преобразования. На выходе генерируется набор тестовых программ, удовлетворяющий критерию полноты. Тестовый набор ранга m полон, если каждый элемент из множества m-ок операторов хотя бы один раз встречается в сгенерированном наборе тестовых программ.

В файле конфигурации содержится формализованное описание условий применения конкретного преобразования. Для преобразования «Разрезание цикла» конфигурация имеет вид:

```
<?xml version="1.0" encoding="UTF-8"?>
<program class="loop_distribution">
  <params … />
  <body>
    <statFor>
      <block m="3" statAssign="1" statIf="1">
        <statIf statAssign="1" />
      </block>
      <block m="3" statAssign="1" statIf="1">
        <statIf statAssign="1" statGoTo="1" />
      </block>
    </statFor>
  </body>
</program>
```

Программы, сгенерированные по данной конфигурации, содержат оператор цикла, тело которого логически состоит из двух блоков. Блоками являются все возможные тройки двух операторов. Количество тестов при этом в полном наборе 64 (два в кубе умножить на два в кубе).

Диалоговый режим анализа и преобразования программ.

Диалоговое распараллеливание является компромиссом между ручным и автоматическим. Не вся информация о программе содержится в коде. Для определения информационных зависимостей в программе могут быть полезны диапазоны значений переменных, которые, как правило, в программе не определены. Побочные эффекты функций, находящихся во внешних библиотеках, также обычно не известны. Но в диалоговом режиме компиляции появляется возможность запросить недостающую информацию у пользователя. Аналогично в диалоге может быть получена информация о размерах массивов данных, о соотношениях между различными переменными, о псевдонимах (alias) переменных.

Преимущества диалогового распараллеливания по сравнению с автоматическим.

- Уточнение информационных зависимостей
- Изменение порядка выполнения операций
- Принятие решений о целесообразности преобразований
- Быстрая адаптируемость к новым архитектурам
- Перебор вариантов распараллеливания

Пример преимущества диалоговой распараллеливающей компиляции.

Алгоритм Флойда имеет на входе матрицу весов графа, а на выходе – матрицу кратчайших расстояний. Может быть использован для построения матрицы достижимостей (транзитивного замыкания) графа, для поиска кратчайших путей между всеми парами вершин, для поиска компонент сильной связности ориентированного графа и для построения фактор-графа по компонентам сильной связности.

```
FOR K=1 TO N DO
 FOR I=1 TO N DO
  FOR J=1 TO N DO
    IF A(I,J)>A(I,K)+A(K,J) THEN
       A(I,J)=A(I,K)+A(K,J)
```

Граф зависимостей может показать наличие циклически порожденных зависимостей.

Следовательно, ни один цикл из данного тройного гнезда не может быть распараллелен. Символьный анализ может показать, что информационные зависимости реализуются при I=K или J=K. Выполнив символьно любую из этих подстановок, можно увидеть, что логическое выражение условного оператора ложно, если элементы матрицы положительны.Но тогда нет обращения программы к генератору A(I,J)= и зависимостей, на самом деле, нет. Следовательно, распараллеливание циклов со счетчиками I,J возможно. Но как компилятор для такого анализа может узнать, что все элементы матрицы положительны? Здесь компилятор мог бы уточнить диапазоны данных в диалоге с пользователем, после чего доказать возможность распараллеливания.

Анализ и уточнение информационных зависимостей.

В основе оптимизирующих и распараллеливающих преобразования программ лежит анализ информационных зависимостей [11]. Не всегда возможно точно определить такие зависимости с помощью традиционных средств (неравенства Банерджи и др.) [12]. Для уточнения зависимостей в ДВОР используются символьное преобразование выражений «линеаризация». Линеаризацией относительно набора переменных ai называется преобразование выражения к виду: e1*a1+e2*a2+…+en*an+en+1, где ei – выражения, ai – переменные, ни одна из которых не входит ни в одно из выражений ei. Линеаризация выражений необходима для реализации следующих проектов ДВОР: «Граф зависимостей по данным»; «Решетчатый граф»; «Распараллеливание рекуррентных циклов»; «Генерация MPI-кода». В линеаризации выражений в качестве базовых переменных кроме счетчиков циклов могут выступать и внешние переменные (которые не меняют своего значения в преобразуемом фрагменте) и имена массивов с различными индексами (при распараллеливании рекуррентных циклов). При линеаризации на этапе компиляции приводятся подобные с учетом типов данных. Например, для выражения i+127 – (j-1) при его линеаризации коэффициенты при переменных имеют вид либо {1,-1,-128} в 8-битной арифметике, либо {1,-1,128} в 16-битной (или более) арифметике.

Анализ указателей – это глобальный анализ программы, позволяющий для каждой точки программы, содержащей указатель, вычислить множество фрагментов памяти, адреса которых может принимать указатель в этой точке программы. Обычно результаты анализа указателей используются для уточнения потока данных [13], в ДВОР анализ указателей используется для уточнения информационных зависимостей. С помощью анализа указателей можно получить новые информационные зависимости, обусловленные наличием в программах разных имен одной и той же ячейки памяти, которые не учитываются такими методами, как НОД тест и

неравенства Банержи, Омега тест и другими, определяющими зависимости между вхождениями переменных. Межпроцедурный анализ указателей [14] В ДВОР реализован независимо от построения SSA-формы.

В рамках проекта ДВОР реализовано построение графа потока управления и графа вызовов подпрограмм. Граф потока управления с вершинами - операторами и дугами – передачами управления между операторами предназначен для использования при анализе программ, в том числе с целью более точного определения информационных зависимостей. Этот граф определяет порядок выполнения операторов программы при ее последовательном выполнении. Граф вызовов подпрограмм, в котором вершины соответствуют подпрограммам, а дуги – их вызовам, используется при межпроцедурном анализе для определения информационных зависимостей между отдельными подпрограммами. Этот граф хранит информацию обо всех вызовах подпрограмм, совершаемых в теле каждой отдельной подпрограммы.

Для выявления зависимостей между парами вхождений необходимо анализировать индексные выражения массивов и выражения для границ циклов, охватывающих данные вхождения. Эти выражения должны быть линейными относительно счетчиков циклов, то есть они должны иметь следующий вид: c1*i1+c2*i2+…+cn*in+c, где i1,i2,…,in счетчики циклов, охватывающих данные вхождения; c,c1,c2,…,cn – целые числа. В противном случае считается, что данные вхождения порождают информационную зависимость. Бывают случаи, когда в анализируемых выражениях присутствуют внешние переменные, но, применяя символьный анализ, иногда удается установить отсутствие зависимости между вхождениями.

Пример.

```
for(int i=0; i<100; i++)
  A[i+N] = A[i+N-1] + B[i];
```

Очевидно, что граф информационных связей для данного примера будет состоять из одной дуги, идущей из вхождения A[i+N] в вхождение A[i+N-1]. Но индексные выражения обоих вхождений являются нелинейными, поэтому если не применять символьный анализ, то анализ информационной зависимости построит граф в котором будут 4 дуги: из вхождения A[i+N] в вхождение `A[i+N-1]`, `из A[i+N-1] в A[i+N]` и две петли самозависимости вхождений `A[i+N] и A[i+N-1]`.

В ДВОР для определения зависимости между парой вхождений реализованы два теста: НОД-тест и тест на основе неравенств Банержи. Указанные методы имеют два основных достоинства: это простота программной реализации и линейная сложность. Существенным недостатком этих методов является их неточность определения информационной зависимости. В некоторых случаях НОД-тест и тест Банержи ошибочно определяют наличие зависимости между парой вхождений. В ДВОР реализован метод уточнения графа информационных связей с помощью элементарных решетчатых графов.

Пример.

```
for(int i=1;i<99;i=i+1)
{
  a[i+1][i]=b[i]+a[i][100-i]+c[i];
}
```

В данном примере при построении графа информационных связей с помощью неравенств Банержи и НОД теста будет неверно определено наличие зависимости a[i][100-i] от a[i+1][i]. Если для построения графа использовать элементарные решетчатые графы, то будет верно установлено ее отсутствие.

Одним из важных требований, предъявляемых при проектировании современных электронных цифровых устройств (на основе ПЛИС или микросхем заказной логики), является малое время разработки. Часто разработчик ради сокращения цикла проектирования устройства готов пожертвовать качественными характеристиками, такими как быстродействие, площадь кристалла, энергопотребление и т. д. Достижению этой цели может способствовать применение средств, позволяющих спроектировать цифровое устройство по описанию алгоритма его работы на языке программирования высокого уровня, такого как C. Существующие в настоящее время немногочисленные программные комплексы, имеющие данную возможность, характеризуются узким классом входных алгоритмов, эффективно реализуемых ими аппаратно, а также высокой стоимостью.

Конвертор языка программирования Си в язык описания электронных схем.

На базе ДВОР разрабатывается генератор электронных схем на языке описания оборудования (HDL). Данная задача включает в себя следующие этапы:

1. Получение внутреннего представления программы.
2. Построение графа вычислений для каждого конвейеризуемого участка программы [15].
3. Генерация HDL-описаний участков электронной схемы, реализующих построенные графы вычислений, во внутреннем представлении программы на HDL
4. Преобразование внутреннего представления результирующей HDL-программы в текст на языке описания оборудования. В настоящее время поддерживается вывод на языке VHDL.

На вход конвертера можно подавать (на момент написания этого текста) одномерные циклы, содержащие операторы if и операторы присваивания с вхождениями переменных, содержащими регулярные линейные индексные выражениями. Предполагается расширение этого ограничения на гнезда циклов.

Генерация параллельного кода.

Для циклов языка ФОРТРАН и аналогичных им циклов языка Си сделано автоматическое определение распараллеливаемости. Из внутреннего представления ДВОР сделан вывод текста на языках Си и ФОРТРАН. Для распараллеливаемых программ разрабатывается автоматический вывод текста с прагмами OpenMP и, для вычислительных систем с распределенной памятью – вызов функций библиотеки MPI. Обсуждается генерация кода для CUDA.

Работа поддержана ФЦП "Научные и научно-педагогические кадры инновационной России", ГК № 02.740.11.0208 от 07.07.2009 г.

ЛИТЕРАТУРА:

1. Открытая распараллеливающая система – http://ops.rsu.ru/
2. LLVM & clang C frontend – http://www.llvm.org/
3. Воеводин В.В. Воеводин Вл.В. Параллельные вычисления, С-Петербург «БХВ-Петербург», 2002, 599 с.
4. Штейнберг Б.Я. Математические методы распараллеливания рекуррентных программных циклов на суперкомпьютеры с параллельной памятью.// Ростов-на-Дону, Издательство Ростовского университета, 2004 г., 192 с.
5. Нис З. Я. Преобразования программ: контроль семантической корректности // Изв. вузов. Сев.-Кавк. регион. Естественные науки. 2010 г., №1. С. 18-21.
6. Muchnick Steven S. Advanced Compiler-Design and Implementation. Morgan Kaufmann Publishers, An Imprint of Elsevier. 1997. – 860 p.
7. J.-F. Collard, D. Barthou, P. Feautrier. Fuzzy Array Dataflow Analysis. ACM SIGPLAN Notices, Volume 30, Issue 8. 1995.
8. Feautrier P. Dataflow analysis of scalar and array references // International Journal of Parallel Programming. V. 20(1). February 1991. P. 23–52.
9. Feautrier P. Array expansion //In ACM Int. Conf. on Suprecomputing, St Malo, 1988.
10. Алымова Е.В. Автоматическая генерация тестов на основе конфигурационных файлов для оптимизирующих преобразований компилятора // Известия вузов. Северо-Кавказский регион. Естеств. науки. № 3, 2010.
11. Серебрянский А.И., Сыч Г.А. Обзор распараллеливающих компиляторов. // Системное программирование. Вып. 3. СПб., 2008. С. 157-177
12. R. Allen, K. Kennedy Optimizing compilers for Mordern Architetures// Morgan Kaufmann Publisher, Academic Press, USA, 2002, 790 p.
13. Альфред В. Ахо, Моника С. Лам, Рави Сети, Джеффри Д. Ульман. Компиляторы: принципы, технологии и инструментарий, 2-е изд.: Пер. с англ. – М.: ООО «И.Д. Вильямс», 2008. – 1184 с.
14. Дроздов А.Ю, Владиславлев В.Е. Межпроцедурный анализ указателей // Информационные технологии. Приложение № 2. 2005.
15. Штейнберг Р. Б. Вычисление задержки между стартами конвейеров с учётом времени пересылки данных. – Труды конференции РАСО-2006.