УДК 004. 4'41

АНАЛИЗ УКАЗАТЕЛЕЙ ДЛЯ РАСПАРАЛЛЕЛИВАНИЯ

С.В. Полуян
Южный федеральный университет
Россия, 344006, г.Ростов-на-Дону, Б.Садовая ул., 105
E-mail: steka@front.ru

Ключевые слова: анализ указателей, информационная зависимость по памяти, распараллеливание
Key words: points-to analysis, memory-based dependence, parallelization

Целью данной работы является создание алгоритма анализа указателей, позволяющего использовать результаты анализа для корректного распараллеливания программ, содержащих указатели. Этот алгоритм программно реализован в «Диалоговом высокоуровневом оптимизирующем распараллеливателе программ».
POINTS-TO ANALYSIS FOR PARALLELIZATION / S.V. Poluyan (Southern Federal University, 105 B Sadovaya-street, Rostov-on-Don, 344006, Russia, E-mail: steka@front.ru). The aim of this work is to create an algorithm points-to analysis that allows use the results of analysis for correct parallelization of programs that use pointers. The analysis is implemented in "High-level dialogue-based optimizing parallelizer" software.

1. Введение

Указатели – один из важнейших элементов таких языков программирования, как Си, Паскаль и других. Они позволяют работать со структурированными данными, динамическими структурами, генерировать вызовы по ссылке и многое другое. Указатели представляют собой переменные, значениями которых являются адреса памяти. Существование в языках программирования таких переменных приводит к тому, что в программах появляются разные имена одной и той же ячейки памяти, что не может не сказаться на возможности автоматически распараллелить такую программу.

Данная работа посвящена применению анализа указателей для уточнения информационных зависимостей [1] программ, написанных на языке Си. Результаты этой работы используются в «Диалоговом высокоуровневом оптимизирующем распараллеливателе» [2].

Особенностью данной работы является использование результатов анализа указателей для уточнения информационных зависимостей, а не потоков данных. Задача уточнения информационных зависимостей имеет большое значение в ДВОР, так как без корректного анализа информационных зависимостей нельзя применять такие распараллеливающие преобразования программ, как перестановка фрагментов кода, переименование переменных, разбиение циклов и так далее.

Для определения информационной зависимости между вхождениями переменных при определенных значениях индексных выражений существуют такие методы, как НОД тест [1] и неравенства Банержи [3], Омега тест [1] и другие. Однако эти методы не учитывают

зависимости, обусловленные наличием в программах разных имен одной и той же ячейки памяти, поэтому к информации о зависимостях в ДВОР добавляются данные, полученные с помощью анализа указателей [4].

Для сбора информации об указателях в ДВОР реализован межпроцедурный анализ указателей, который имеет следующие особенности:

- позволяет получить результаты для исходной, а не преобразованной (например, к SSA-форме) программы, так как для анализа информационных зависимостей нужна информация об указателях для исходной программы.
- является универсальным для разных языков программирования, так как реализован на мультиязыковом внутреннем представлении называемом в ДВОР Reprise [5], в узлах которого содержатся высокоуровневые конструкции языков программирования, а его структура не зависит от входного языка.
- уделено внимание простоте обновления информации об указателях при применении преобразований программ.

Алгоритмы анализа указателей, реализованные в ДВОР, можно разделить на два типа: решающие внутрипроцедурную и межпроцедурную части задачи анализа указателей.

2. Внутрипроцедурная часть анализа указателей

Внутрипроцедурная часть анализа основана на обходе графа потока управления [6]. Вершинами графа потока управления в ДВОР являются все операторы фрагмента программы, а не базовые блоки, что позволяет ускорить и упростить обновление графа при трансформациях фрагмента кода в процессе применения преобразований программ.

Во внутрипроцедурной части анализа используется итерационный алгоритм, начинающий свою работу с первой вершины графа потока управления. Разбирая узел графа, алгоритм определяет места записей в указатели и вычисляет их значения. Чтобы вычислить значения указателей, находящихся внутри циклов, строится дерево вложенности циклов, в узлах которого содержится информация о циклах, а в листьях находятся выражения (рис. 1).

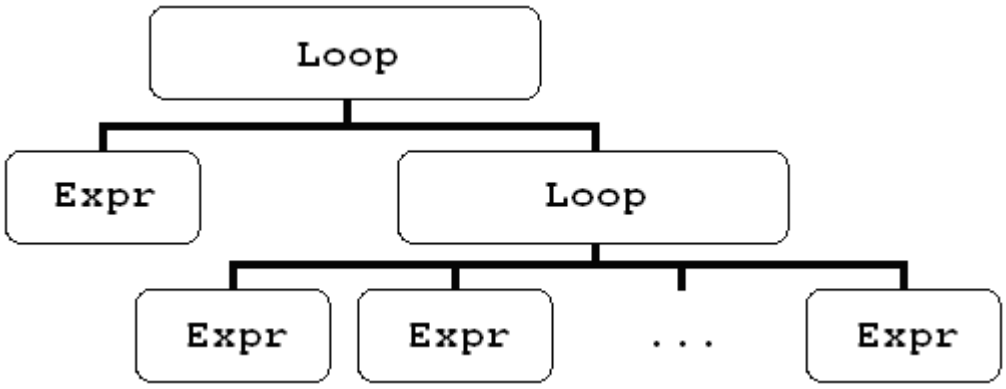

Рис. 1. Дерево вложенности циклов.

Далее вычисленные значения распространяются по графу потока управления. Для оптимизации этого процесса используется дерево доминаторов [7]. Это дерево позволяет быстро найти все узлы, достижимые из вычисленного узла.

В качестве примера рассмотрим следующий фрагмент программы на Си:

```
q=&m[2];
if (*q!=0)
   q++;
else
   q=m;
p=q;
```

Граф потока управления и дерево доминаторов для этого фрагмента программы изображены на рис. 2.

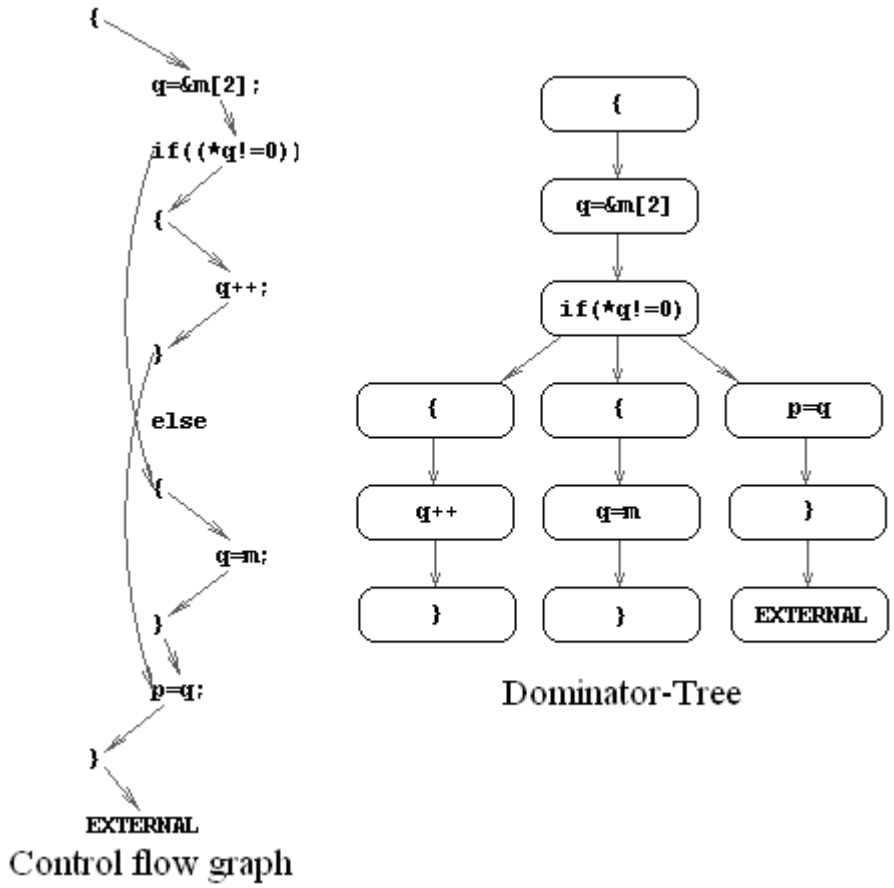

Рис. 2. Граф потока управления и дерево доминаторов.

Таким образом, на следующей итерации алгоритма вся информация, необходимая для вычисления значений указателей, уже будет содержаться в рассматриваемом узле графа.

В итоге, итерационный алгоритм завершит свою работу после того, как во всех узлах графа значения указателей окажутся вычисленными.

Для сбора информации о псевдонимах используется понятие абстрактной ячейки памяти [8]. Абстрактная ячейка памяти – это поименованная область памяти, к которой возможен доступ из программы. В качестве представления абстрактной ячейки памяти было выбрано:

<Имя, Смещение>,

где «Имя» – имя переменной, а «Смещение» – смещение в памяти.

На основе понятия абстрактной ячейки памяти реализован механизм анализа псевдонимов при использовании арифметических операций с указателями. При определении псевдонима механизм автоматически учитывает размер типа величин, адресуемых указателями. И хотя арифметические операции с указателями имеют смысл в основном при работе со структурными данными такими, как массивы, реализован механизм, работающий со всем множеством абстрактных ячеек памяти программы.

Для хранения значений указателей на внутрипроцедурном уровне в ДВОР реализована хэш-таблица вхождений указателей. Эта таблица содержит следующие поля:

<Вхождение указателя, Множество абстрактных ячеек памяти>

Таким образом, с каждой переменной указательного типа связано множество абстрактных ячеек памяти, на которые может указывать данная переменная.

3. Межпроцедурная часть анализа указателей

В межпроцедурной части анализа используются чувствительный [9] и нечувствительный [10] к контексту алгоритмы. Контекстно-чувствительный алгоритм позволяет провести анализ указателей для каждого вызова процедуры с учетом контекста этого вызова, что существенно повышает точность анализа, но этот алгоритм является достаточно долгим. Для ускорения анализа используются шаблоны, позволяющие при необходимости применять нечувствительную к контексту версию алгоритма.

Для передачи параметров в процедуру используются записи активаций [11]. Значимыми при анализе указателей полями записей активации являются следующие поля:

- Фактические параметры – список фактических параметров процедуры указательного типа с множеством значений.
- Возвращаемое значение – множество значений указателя для результата, имеющего указательный тип.
- Связь по данным – значения нелокальных переменных указательного типа на момент вызова процедуры.

На межпроцедурном уровне информация об указателях связывается с соответствующим узлом графа вызовов [11]. Пример графа вызовов, автоматически построенного в ДВОР, приведен на рис. 3.

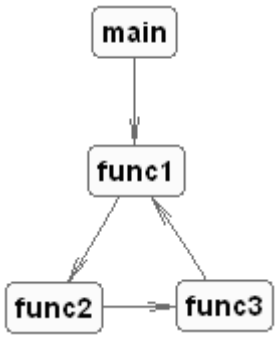

Рис. 3. Граф вызовов.

4. Применение анализа указателей

Описанный выше алгоритм анализа указателей позволяет определить для каждого вхождения указателя множество возможных значений этого указателя. С помощью этой информации можно найти новые информационные зависимости, которые могут существовать между вхождением разыменнованного указателя и вхождением переменной или между вхождениями разыменнованых указателей.

Рассмотрим пример поиска информационных зависимостей, обусловленных наличием в программе указателей:

```
int *a, *b, *c;
 ...
c = a;
 ...
*a=*b;
*b=*c;
```

Не используя результаты анализа указателей можно сразу определить наличие зависимости между вхождениями переменной b. Далее для поиска новых информационных зависимостей необходимо проанализировать множества возможных значений указателей на пересечение. При этом, если множества возможных значений двух вхождений указателей

пересекаются, то между этими вхождениями может существовать зависимость. В данном примере можно видеть, что переменные a и c указывают на одну и ту же ячейку памяти, следовательно, между вхождениями *a и *c тоже существует зависимость.

5. Заключение

На данный момент разработано множество преобразований программ для распараллеливающих компиляторов, но часть из них способна корректно работать лишь с переменными элементарных типов данных, в то время как появление указателей в программах делает эти преобразования способными нарушить корректность программ. Описанный в данной работе алгоритм уточнения информационных зависимостей позволяет не допустить некорректное применение таких преобразований.

Список литературы

1. Allen R., Kennedy K. Optimizing compilers for Mordern Architetures // Morgan Kaufmann Publisher, Academic Press, USA, 2002, 790 p.
2. Диалоговый высокоуровневый оптимизирующий распараллеливатель программ (ДВОР). URL: http://www.ops.rsu.ru.
3. Banerjee U. Data dependence in ordinary programs. — Urbana, 1976. — (Tech. Rep. / Univ.III; 76–837).
4. Дроздов А.Ю, Владиславлев В.Е. Межпроцедурный анализ указателей // Информационные технологии. Приложение № 2. 2005.
5. Петренко В.В. Внутреннее представление REPRISE распаралелливающей системы // PACO'2008, Труды четвертой международной конференции «Параллельные вычисления и задачи управления», Москва.: 27-29 октября 2008 г.
6. Нис З.Я. Анализ потока управления в открытой распараллеливающей системе // Искусственный интеллект. 2005. № 3. с. 461 – 464.
7. Свами М., Тхуласираман К. Графы, сети и алгоритмы. М: Мир, 1984. 455с.
8. Yair Sade, Mooly Sagiv, and Ran Shaham. Optimizing C multithreaded memory management using thread-local storage. In Compiler Construction, pages 137-155, 2005.
9. John Whaley, Monica S. Lam. Cloning-based context-sensitive pointer alias analysis using binary decision diagrams // In SIGPLAN Conference on Programming Language Design and Implementation, 2004.
10. Erik Ruf. Context-insensitive alias analysis reconsidered // In Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation, pages 13-31, June 1995.
11. Ахо, Альфред В., Лам, Моника С., Сети, Равви, Ульман, Джеффри Д. Компиляторы: принципы, технологии и инструментарий, 2-е изд.: Пер. с англ. – М.: ООО «И.Д. Вильямс», 2008. – 1184 с.