УДК 681.3

З.Я. Нис
Ростовский государственный университет, г. Ростов-на-Дону, Россия
irishrover@mail.ru

Анализ потока управления в открытой распараллеливающей системе

В данной работе рассмотрены средства анализа потока управления в открытой распараллеливающей системе, основанные на применении графа управления и графа вызовов подпрограмм.

В компиляторах и других системах преобразований программ для анализа и оптимизации потока управления традиционно используются граф управления и граф вызовов подпрограмм.

Напомним, что графом управления программы (control flow graph) называют ориентированный граф, вершинами которого являются операторы программы (либо базовые блоки программы), а дуга исходит из вершины A в вершину B тогда и только тогда, когда возможна непосредственная передача управления от оператора A к оператору B.

Графом вызовов подпрограмм (call graph) называется ориентированный граф, вершинами которого являются все подпрограммы программы, а дуга соединяет вершины A и B тогда и только тогда, когда в подпрограмме A существует явный вызов подпрограммы B.

С помощью графа управления можно определять недостижимые фрагменты кода, некоторые виды зацикливания программ, возможность перегруппировки операторов для использования возможностей процессора по оптимизации кэширования памяти; также граф управления может использоваться при контроле корректности оптимизирующих преобразований и при внутрипроцедурном (intraprocedural) анализе. Ещё одно применение графа управления – этап вставки φ-функций при построении статических форм с однократным присваиванием (SSA-forms) [1].

Граф вызовов подпрограмм, в свою очередь, позволяет обнаруживать и исключать из процесса связывания (linking) невызываемые подпрограммы, обнаруживать рекурсивные функции и оптимизировать их вызовы, делать выводы о возможности использования функций как подставляемых (inlining), а также помогает при проведении межпроцедурного (interprocedural) анализа.

Оба графа широко используются в таких известных программных продуктах, как оптимизирующие компиляторы gcc [2], система преобразований «текст-в-текст» Cetus (университет Purdue) [3], пакет SUIF (Стэндфордский университет) [4], распараллеливающая и оптимизирующая система Polaris (университет Иллинойс) [5], [6], система PIPS (строится и используется граф управления) [7], пакет V-Ray (строится и используется в расширенном аналоге графа управления – иерархическом графе управления) [8].

Граф вызовов подпрограмм и граф управления также используются и в открытой распараллеливающей системе (ОРС) [9], проекте, разрабатываемом на кафедре алгебры и дискретной математики Ростовского государственного университета группой студентов и аспирантов под руководством Б.Я. Штейнберга. ОРС

предназначена для автоматического и полуавтоматического распараллеливания программ с процедурных языков программирования (Фортран, Паскаль, Си) на параллельные компьютеры, ориентированные на математические вычисления. ОРС содержит парсеры с этих языков во внутреннее представление – древовидную систему связанных объектов различных классов C++, представляющих программу. Также в состав ОРС входит библиотека преобразований программ.

Графы управления и вызовов также реализованы в виде классов и являются основным либо вспомогательным средством при проведении некоторых преобразований. Оба графа строятся в ОРС по внутреннему представлению программы. Графы реализованы как классы языка C++ (языка, на котором разработана вся система ОРС, включая библиотеку преобразований) и включают методы построения по заданному внутреннему представлению, а также методы доступа к отдельным вершинам и дугам. В качестве вершин графа выступают объекты внутреннего представления разобранной парсером программы. Благодаря использованию независимого от языка внутреннего представления, графы управления и вызовов подпрограмм во внутреннем представлении также оказываются не зависящими от языка. Это, в свою очередь, позволяет сделать не зависящими от языка проверки корректности преобразований или даже целые преобразования. В частности, это облегчит внесение изменений во входной язык компилирующих продуктов, сделанных на основе ОРС.

В отличие от некоторых других работ, предполагающих, что граф управления в качестве вершин имеет базовые блоки (линейные участки программы), в ОРС применен несколько иной подход, при котором вершинами являются все операторы фрагмента программы, для которого строится граф. Это позволяет несколько упростить и ускорить обновление графа при трансформациях фрагмента кода, для которого граф строился, и смежных фрагментов. Граф управления в ОРС хранится в памяти в виде ассоциативного массива, в котором ключами являются все операторы фрагмента, а значениями – списки операторов, непосредственно достижимых из данного.

Одно из основных применений графа управления в ОРС связано с обнаружением всех входов и выходов для заданного фрагмента кода. Это часто требуется при проверке корректности и применимости преобразований. Приведем несколько примеров.

Широко используемое в оптимизирующих компиляторах преобразование «подстановка вперёд» и его частный случай «протягивание констант». Суть этого преобразования заключается в том, что правая часть оператора присваивания подставляется вместо последующих вхождений левой части этого же оператора. Это преобразование может помочь избавиться от некоторых дуг информационных зависимостей и облегчить распараллеливание. Одним из необходимых условий применимости данного преобразования является наличие только одного входа во фрагмент. Таким образом, следующий фрагмент кода:

```
…
        A=5
LAB1: X=A
…
```

не эквивалентен фрагменту

```
…
A=5
LAB1: X=5
…
```

как раз из-за наличия второго входа в оператор, помеченный LAB1.

Аналогичная ситуация имеет место и в преобразовании «перестановка двух операторов фрагментов», которое применяется, например, при оптимизации использования кэш-памяти и при вычислениях на VLIW [10]. Например, если поменять местами следующие два оператора фрагмента:

```
LAB1:  X=E1
       Y=E2,
```

чтобы получилось

```
       Y=E2
LAB1:  X=E1,
```

то новый фрагмент не будет семантически эквивалентен исходному.

Преобразование «слияние двух циклов», применяемое при оптимизации кода при последовательном и конвейерном выполнении, заменяет фрагмент программы, состоящий из двух смежных циклов с одинаковыми заголовками, одним циклом. Точно так же, как и в предыдущих случаях, наличие «лишнего» входа в тело одного из сливаемых циклов не позволит провести преобразование, так как получится код, неэквивалентный исходному. Код

```
        DO I=1,10
        A(I)=E1(I)
       END DO
LAB1:  DO J=1,10
       B(J)=E2(J)
       END DO,
```

не эквивалентен коду

```
LAB1:  DO I=1,10
       A(I)=E1(I)
B(I)=E2(I)
       END DO.
```

В ОРС на основе графа управления построены и другие преобразования. Например, «обнаружение и удаление недостижимого кода», в котором по графу управления для фрагмента программы строится список операторов фрагмента, недостижимых из «точки входа» во фрагмент. Эти «недостижимые» операторы могут быть при необходимости удалены из фрагмента без изменения семантики его работы.

В ближайшее время предполагается использование графа управления при проведении внутрипроцедурного (intraprocedural) анализа. Также граф управления предполагается использовать для проверки корректности преобразований при их автоматическом порождении и для генерации SSA-форм.

Граф вызовов подпрограмм в ОРС укладывается в классическую схему, при которой вершинами графа являются все подпрограммы программы или её фрагмента, а дуги ведут от вызывающей подпрограммы к вызываемой. В ОРС граф вызовов также хранится в виде ассоциативного массива, где ключом является имя подпрограммы, а значением – ассоциативный массив, связывающий имена вызываемых из подпрограммы-ключа подпрограмм со списком выражений, в которых производятся эти вызовы.

В настоящее время граф вызовов подпрограмм используется в преобразовании «обнаружение невызываемых подпрограмм», «подстановка тела функции в точку вызова» (inlining), а также ведутся работы по его использованию при осуществлении межпроцедурного анализа.

Литература

1. Cytron R., Ferrante J., Rosen B., Wegman M., Zadeck K.. Efficiently Computing Static Single Assignment Form and the Control Dependence Graph // ACM Transactions on Programming Languages and Systems. – 1991. –Vol. 13 (4). – P. 451-490.
2. http://gcc.gnu.org/
3. http://min.ecn.purdue.edu/~troyj/papers/johnson04cetus.pdf
4. Wilson R., French R., Wilson Ch., Amarasinghe S., Anderson J., Tjiang S., Liao S., Tseng C., Hall M., M. Lam, Hennessy J. SUIF: An Infrastructure for Research on Parallelizing and Optimizing Compilers.
5. Faigin K., Hoeflinger J., Padua D., Petersen P., Weatherford S. The Polaris Internal Representation.
6. Paek Y., Petersen P. A Data Dependence Graph in Polaris.
7. http://www.cri.ensmp.fr/~pips/
8. http://parallel.ru/v-ray/
9. Штейнберг Б.Я. Открытая распараллеливающая система // Труды междунар. конф. «Параллельные вычисления и задачи управления» (PACO'2001). – М.: ИПУ РАН. – С. 214-220.
10. Штейнберг Б.Я. Математические методы распараллеливания рекуррентных циклов для суперкомпьютеров с параллельной памятью. – Ростов н/Д: Изд-во Рост. ун-та, 2004. – 192 с.

З.Я. Ніс
Аналіз потоку керування у відкритій распаралелювальній системі
У даній роботі розглядаються засоби аналізу потоку керування у відкритій распаралелювальній системі, засновані на застосуванні графа керування і графа викликів підпрограм.

Z.Ya. Nis
Control Flow Analysis in Open Parallelizing System
This work deals with methods of Control Flow analysis in the Open Parallelizing System, based on usage of Control Flow graph and Call Graph.

Статья поступила в редакцию 04.07.2005.