С.А. Логачева

АНАЛИЗ ЗАВИСИМОСТЕЙ ПО ДАННЫМ НА БАЗЕ АЛГОРИТМА ШОСТАКА[1]

1. ВВЕДЕНИЕ

При распараллеливании программы одной из основных проблем является выявление зависимости по данным. Существует три типа зависимостей: потоковая, антизависимость и выходная. Потоковая является истинной, остальные — «ложными». Далее мы будем говорить только об истинных зависимостях. Для их выявления существует ряд приближенных тестов, основанных на различных математических фактах.

В настоящей работе строится точный тест для анализа зависимостей по данным, основанный на исследовании разрешимости системы линейных неравенств с двумя переменными (полученной в результате выявления зависимостей) на базе алгоритма Шостака [1]. Тест состоит из двух этапов. Начальный этап включает в себя подбор пар операторов, “подозрительных” на зависимость. Второй этап состоит в применении модифицированного алгоритма Шостака.

В п. 2 данной работы приводится модель входной программы; в п. 3 описывается алгоритм Шостака и его модификация; в п. 4 производится сравнение полученных результатов алгоритма Шостака с результатами наиболее известных тестов на зависимость; в п. 5 описывается программная реализация проекта.

Все понятия, не определяемые в этой работе, могут быть найдены в [2].

2. МОДЕЛЬ ПРОГРАММЫ

Рассматриваются зависимости, порождаемые ссылками на элементы массива, имеющиеся в теле DO-цикла, поскольку при анализе зависимостей по данным решение этой задачи представляет основную трудность.

[1] Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

Предполагается, что не существует смешивания имен, таким образом, ячейки памяти, к которым обращаются по разным именам, являются различными.

В качестве модели программы рассматривается совершенное гнездо циклов (тело циклов находится внутри самого внутреннего цикла) таких, что приращения индексных переменных равны 1 и по каждой размерности используется только одна, отличная от других индексная переменная.

Наконец, предполагается, что индексные выражения линейные с целочисленными коэффициентами, а границы DO-циклов известные константы. (Введенные ограничения обеспечивают точность теста.)

Границы DO-циклов также могут линейно зависеть от индексной переменной любого внешнего цикла. Но в этом случае не всегда можно гарантировать точность теста, поскольку этот вопрос изучен не до конца.

3. АЛГОРИТМ ШОСТАКА

3.1. Определения

Пусть S — множество линейных неравенств вида $ax + by \le c$, где x, y — действительные переменные и a, b, c — действительные коэффициенты. Введем специальную нулевую переменную v_0 и будем считать, что она появляется в S только с нулевым коэффициентом. Таким образом, если неравенство содержит только одну переменную, то второй переменной для него будет v_0.

Построим для S неориентированный мультиграф G следующим образом: каждой переменной, встречающейся в S, сопоставим вершину в G, каждому неравенству — ребро, помеченное этим неравенством, например: ребро, помеченное неравенством $ax + by \le c$, связывает вершины x и y. Назовем

$G = G(S)$ графом для S.

Пусть P — путь в G, заданный последовательностью вершин $v_1, v_2, \ldots, v_{n+1}$ и последовательностью ребер $e_1, e_2, \ldots, e_n$, где $n \ge 1$. Назовем *последовательностью триплетов* последовательность вида $(a_1, b_1, c_1), (a_2, b_2, c_2), (a_n, b_n, c_n)$, где для любого i, $1 \le i \le n$, ребро e_i, помечено неравенством $a_i v_i + b_i v_{i+1} \le c_i$. Путь P является *допустимым*, если b_i и a_{i+1} имеют противоположные знаки для $1 \le i \le n$-1.

Понятно, что допустимые пути соответствуют последовательностям неравенств, которые образованы транзитивными цепями. Например, последовательность

$$x \le y, \quad y \le z, \quad z \le 3$$

образует допустимый путь, состоящий из последовательности вершин x, y, z и последовательности ребер $x - y \le 0$, $y - z \le 0$, $z + 0v_0 \le 3$, а последовательность

$$x \le y, \quad y \le z, \quad -z \le r$$

не образует допустимый путь, так как коэффициенты при z имеют одинаковые знаки.

Путь является *циклом*, если его первая и последняя вершины совпадают. Цикл называется *простым*, если все его промежуточные вершины различны.

Циклические перестановки цикла P являются *допустимыми* тогда и только тогда, когда a_1 и b_n имеют противоположные знаки, где (a_1, b_1, c_1), …, (a_n, b_n, c_n) — последовательность триплетов для P. В этом случае говорим, что цикл P — *перестановочный.*

Заметим, что допустимый цикл с начальной вершиной v_0 не является перестановочным, так как v_0 содержится в S только с нулевым коэффициентом.

Определим *остаточное неравенство* для допустимого пути P как неравенство, полученное с помощью последовательного применения транзитивности к неравенствам, помечающим ребра P. Например, если P образован цепочкой неравенств

$$x \le 2y + 1, \; y \le 2 - 3z, \; -z \le w,$$

то получим

$$x \le 2y + 1 \le 2(2 - 3z) + 1 \le 2(2 + 3w) + 1 = 6w + 5.$$

Таким образом, остаточное неравенство для P имеет вид $x - 6w \le 5$.

Дадим точное определение. *Остатком* r_P для пути P называется триплет (a_P, b_P, c_P), полученный следующим образом:

$$(a_P, b_P, c_P) = (a_1, b_1, c_1) * (a_2, b_2, c_2) * \ldots * (a_n, b_n, c_n),$$

где (a_1, b_1, c_1), …, (a_n, b_n, c_n) — последовательность триплетов для пути P и $*$ — бинарная операция:

$$(a, b, c) * (a', b', c') = (kaa', -kbb', k(ca' - c'b)) \qquad \text{и} \qquad k = \operatorname{sign}(a').$$

Остаточное неравенство для пути P есть $a_P x + b_P y \le c_P$, где x и y соответственно первая и последняя вершины пути P.

Достаточно просто показать, что операция $*$ ассоциативна, поэтому остаточное неравенство определено однозначно.

Индукцией по длине P легко показать, что любая точка (действительное число), удовлетворяющая неравенствам, помечающим ребра пути P, также удовлетворяет остаточному неравенству для P.

3.2. Алгоритм проверки разрешимости множества линейных неравенств с двумя переменными

Пусть P — цикл с начальной вершиной x, тогда точка, удовлетворяющая неравенствам вдоль P, должна удовлетворять остаточному неравенству $a_P x + b_P x \le c_P$. Если $a_P + b_P = 0$ и $c_P < 0$, то остаточное неравенство для P неверно, и мы говорим, что P — *невыполнимый цикл.*

Это значит, что множество неравенств S неразрешимо, если граф G для S имеет невыполнимый цикл. Обратное утверждение верно не всегда. Например, на рис. 1 показан граф G для $S = \{x \le y,\ 2x + y \le 1,\ z \le x,\ w \le z,\ z \le 1 + w,\ z \ge ½\}$, где S является неразрешимым, но граф не имеет невыполнимых циклов.

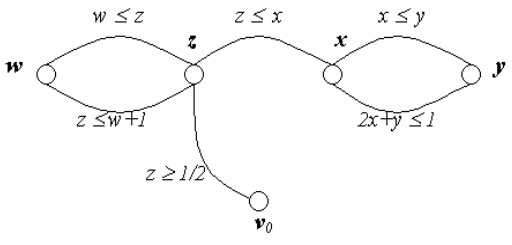

Рис. 1

Основная идея состоит в том, чтобы преобразовать граф G в граф G', который имел бы простой невыполнимый цикл тогда и только тогда, когда множество S было бы неразрешимо.

Пусть G — граф для S. Построим граф G' путем добавлением к графу G для каждого простого допустимого цикла P из G новой дуги, помеченной остаточным неравенством для P. Полученный граф назовем замыканием графа G.

Заметим, что замыкание не обязательно является единственным, так как начальная вершина перестановочного цикла может быть выбрана произвольным образом.

Теорема. *S является неразрешимым тогда и только тогда, когда G′ имеет простой невыполнимый цикл. [1]*

На рис. 2 показано единственное замыкание графа, изображенного на рис. 1. Заметим, что только *xyx*-цикл графа *G* добавляет ребро в *G′*. Цикл v_0xzv_0 графа *G′* является невыполнимым (имеет остаток (0, 0, –1/3)), значит, по теореме множество *S* неразрешимо.

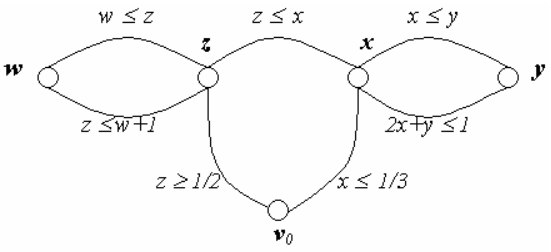

Рис. 2

Таким образом, имеем следующий **алгоритм проверки разрешимости множества неравенств *S***:

1) просматриваем все простые допустимые циклы и их циклические перестановки и вычисляем их остатки. Если находим невыполнимый цикл, тогда *S* неразрешимо;
2) в противном случае строим замыкание графа *G* добавлением новых ребер, помеченных остаточными неравенствами. Затем вычисляем остатки всех вновь сформированных простых допустимых циклов. Если находим невыполнимый цикл, тогда *S* неразрешимо. Иначе *S* имеет решение.

Если *S* является разрешимым, то построение решения приводится в [1]. Заметим, что новые допустимые циклы, образованные в п. 2), должны иметь начальную вершину v_0.

Алгоритм Шостака может быть обобщен для множества строгих неравенств и для множества неравенств с произвольным числом переменных [1].

3.3. Эффективность алгоритма

Метод Шостака требует нахождения простых циклов, для чего существуют несколько алгоритмов (Jonson, Szwarcfiter and Lauer [6], Read and Tarjan [7], [1]) со сложностью по времени $l(|V| + |E|)$ и сложностью по размерности $|V| + |E|$, где l — число порожденных циклов. Эти алгоритмы легко преобразуются для нахождения только простых допустимых циклов. Сложность полученных алгоритмов отличается от сложности исходных на константный множитель, так как каждый цикл имеет длину порядка $|V|$.

Заметим, что множество неравенств с целочисленными константами целочисленно разрешимо, если оно разрешимо в действительных переменных. Обратно не верно. Таким образом, алгоритм Шостака полезный, но не полный тест для проверки целочисленной разрешимости.

3.4. Модификация алгоритма

Для проверки разрешимости системы неравенств в целочисленной области все неравенства с одной переменной должны быть редуцированы путем деления свободного члена на коэффициент при переменной и взятия целой части, например, $10x \le 25$ преобразуется в $x \le 2$, т. е., просматривая данное множество линейных неравенств, редуцируем все неравенства с одной переменной и далее на каждом шаге алгоритма редуцируем полученные остаточные неравенства.

В данной работе из-за введенных ограничений на модель программы, таких, что по каждой размерности используется только одна индексная переменная, отличная от других, множества неравенств будут иметь три различных переменных, одна из которых будет v_0. Таким образом, построенный граф будет иметь три вершины и, соответственно, максимальный размер циклов будет равен трем.

Очевидно, что процесс редуцирования неравенств даст ответ на существование целочисленного решения. Пусть граф состоит из вершин x, y, v_0. Возможны три случая:

1. Если тестируется цикл (xyx), то создается остаточное неравенство, которое редуцируется, тем самым мы попадаем в целочисленную область.
2. Если тестируется цикл (v_0xv_0) или (v_0yv_0), то целочисленная разрешимость очевидна.

3. Если тестируется цикл (v_0xyv_0) (пусть он дан последовательностью неравенств $x \le c_1$, $ax+by \le c_2$, $y \le c_3$), то целочисленная разрешимость легко проверяется графически:

   а) пересечение соответствующих прямых образует треугольную область. Если существует вещественное решение, то существует и целочисленное, например точка $A=(c_1, c_3)$ на рис.3;

   б) в остальных случаях область решения содержит бесконечное число целочисленных точек (например, см. рис.4).

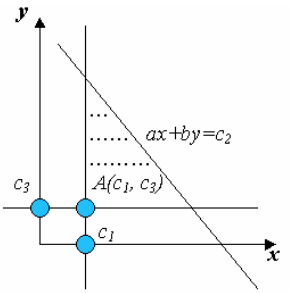

Рис. 3

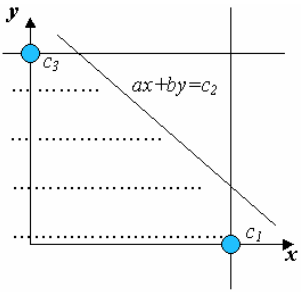

Рис. 4

Таким образом, модифицированный алгоритм Шостака при данных начальных ограничениях обеспечивает точную проверку целочисленной разрешимости для множества линейных неравенств с двумя переменными.

4. СРАВНЕНИЕ РЕЗУЛЬТАТОВ

Проведем сравнение результатов алгоритма Шостака с результатами наиболее известных алгоритмов, таких как НОД-тест, тест Банержи-Вольфа.

Рассмотрим пример:

```
for  (i=1; i<=25; i++)
{
     A[18i-26] = i+5;
     C[i]=A[-i+200];
}
```

Уравнение зависимости выглядит как

$$18x + y = 226$$

при следующих ограничениях:

$$\{x \geq 1,\ x \leq 25,\ y \geq 1,\ y \leq 25\}.$$

Представляя уравнение зависимости в виде двух неравенств, получаем систему линейных неравенств:

$$S = \{18x + y \leq 226,\ 18x + y \geq 226,\ x \geq 1,\ x \leq 25,\ y \geq 1,\ y \leq 25\}.$$

На рис.5 показан граф для S (только сплошные ребра). Тестирование графа указывает на наличие зависимости. Будем искать зависимость с использованием вектора направления (∗). Переход от (∗) к (<) добавляет неравенство $x < y$, которое преобразуется в $x \leq y - 1$ для целых x и y. Штрихованные ребра в графе представляют неравенства, которые выводятся из системы неравенств после добавления ограничения (<), например: неравенство x ≤ 11 получено из этого ограничения и уравнения зависимости.

Теперь рассмотрим ребра в следующем порядке: $-18x - y \leq -226,\ y \leq 25,\ x \leq 11$. Это простой допустимый цикл. Получаем остаточное неравенство $0 \leq -3$, которое является неверным, следовательно, цикл не выполним. Таким образом, зависимости в направлении (<) не существует. Однако, например, тест Банержи-Вольфа[3] не показал независимость в этом направлении, потому что его уравнение зависимости имеет рациональное решение внутри итерационного пространства, (рис.6).

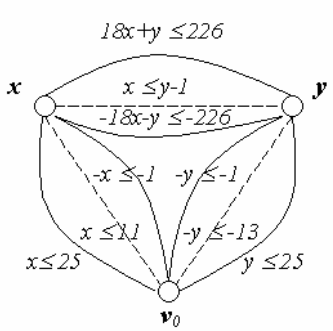

Рис. 5

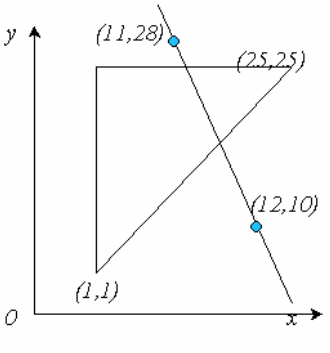

Рис. 6

Также не существует зависимости в направлении (=), значит, она должна быть в направлении (>). Если добавить соответствующие ограничение в граф (только сплошные ребра), то замыкание графа не будет содержать невыполнимые циклы.

Н.О.Д.-тест на этом примере показал зависимость, поскольку НОД(18, 1) = 1 и 226 делится на единицу. Н.О.Д.-тест, конечно, более быстрый тест, чем алгоритм Шостака, но на практике неэффективен, поскольку в большинстве случаев, как и в приведенном выше примере, НОД(a, b) = 1 и тест выдает зависимость там, где ее нет.

Аналогично тест Банержи показал зависимость по всем направлениям, (хотя она существует только в направлении (>)), потому что существуют вещественные решения диофантова уравнения в области итерационного пространства.

Рассмотрим еще один пример:

```
for(i=1; i≤10; i++)
  for(j=1; j≤i-1; j++)
  {
      A[i][j] = A[j][i];
  }
```

Заметим, что ни тест Банержи, ни Н.О.Д.-тест не позволяют вычислить зависимости с удовлетворительной точностью в треугольных циклах. Однако алгоритм Шостака позволяет находить зависимости в циклах такого вида, поскольку основан на проверке разрешимости системы линейных неравенств, например, в данном случае он показал, что зависимости нет.

Таким образом, алгоритм Шостака имеет следующие достоинства:

1) является точным тестом для выявления зависимости по данным при определенных начальных условиях, в отличие от других приближенных тестов, которые могут находить зависимость по данным там, где ее нет на самом деле;
2) позволяет тестировать циклы треугольного вида, что не удается делать многим другим тестам, хотя в данном случае не всегда можно гарантировать точность теста.

Стоит отметить, что алгоритм Шостака позволяет проконтролировать наличие промежуточных записей в ячейку памяти, относительно которой решается задача выявления зависимости (memory-based-dependence и value-based-dependence), что требует использования пресбургеровских формул [5] и представляет интерес для дальнейших разработок.

5. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

Проект реализован на языке C++ с использованием библиотеки MFC и рассчитан на анализ С-программы с определенными ограничениями на синтаксис языка. Он состоит из двух этапов. Первый включает в себя подбор пар операторов, “подозрительных” на зависимость, второй состоит в применении модифицированного алгоритма Шостака. В проект также включены НОД-тест и тест Банержи для возможности сравнения результатов.

5.1. Синтаксис программы

Поскольку в качестве модели входной программы рассматривается совершенное гнездо циклов с определенными ограничениями, используется усеченный синтаксис.

Реализованы стандартные типы: int, float, char; стандартные арифметические операции; операторы: присваивания и for-цикла. Программа должна состоять из блока описаний и процедуры main. В блоке описаний должны быть описаны все переменные, а процедура main должна состоять из последовательности операторов for-цикла и операторов присваивания.

5.2. Подбор пар операторов, “подозрительных” на зависимость

Текст входной программы переводится в дерево внутреннего представления. Каждая вершина дерева имеет определенный тип (оператор, переменная, выражение), информационные поля (количество и значения которых зависят от типа), списки непосредственных потомков и предшественников.

Для каждой вершины оператора присваивания строится множество входных и выходных переменных. Далее происходит обход по дереву внутреннего представления, рассматривается вершина оператора присваивания и сравнивается множество входных и выходных переменных данной вершины с множествами входных и выходных переменных всех потомков данной вершины, которые являются операторами присваивания. В зависимости от пересечения этих множеств получаются пары операторов, “подозрительных” на зависимость, антизависимость или выходную зависимость.

Если встречается переприсваивание переменной, т. е. возникает выходная зависимость, то сравниваются индексные выражения массивов (по которым возникает выходная зависимость) в обоих операторах. Если индексные выражения совпадают, то происходит прерывание просмотра по этой ветке в данной вершине, что частично обеспечивает value-based dependence, например:

```
for (I=1; I<100; I++)
{
S1:   A[I]=…;
      …
S2:   A[I]=…;
S3:   …=A[I+1];
}
```

В данном случае зависимости между S1и S3 не существует, так как происходит переприсваивание переменной в операторе S2. И тест не выдаст пару S1 и S3 в качестве операторов, “подозрительных” на зависимость, потому что для оператора S1 произойдет прерывание просмотра в вершине оператора S2.

После того как произошел подбор пар операторов, “подозрительных” на зависимость, происходит разбор индексных выражений массивов, порождающих зависимость, и построение уравнения зависимости.

В случае гнезда циклов каждая размерность тестируется отдельно, что не нарушает точности теста, поскольку при данной модели программы граф неравенств в алгоритме Шостака распадается на отдельные графы, у которых общей является только одна вершина v_0. Таким образом, графы можно исследовать независимо друг от друга, а значит, можно применять тест к каждой размерности отдельно. Если хотя бы по одной размерности тест показал независимость, то и система уравнений зависимости не имеет решения, следовательно, зависимости в целом не существует.

5.3. Реализация алгоритма Шостака

Уравнение зависимости и соответствующие ограничения на индексные переменные преобразуются в систему линейных неравенств, поскольку алгоритм Шостака основан на построении неориентированного мультиграфа для системы линейных неравенств. Для нахождения всех простых циклов

графа был использован алгоритм Szwarcfiter and Lauer [6]. Граф задается массивом вершин, каждая из которых имеет список инцидентных ей ребер.

6. ЗАКЛЮЧЕНИЕ

Практическим результатом данной работы является построение точного теста для анализа зависимости по данным, написанного на языке C++ с использованием библиотеки MFC, на вход которого подается текст С-программы с усеченным синтаксисом: программа состоит из блока описаний и процедуры main. Процедура main содержит совершенное гнездо циклов, у которых границами являются известные константы; приращение индексной переменной равно единице; по каждой размерности используется только одна индексная переменная. (Эти ограничения обеспечивают точность теста.) Следует отметить, что тест работает и тогда, когда границы индексных переменных являются линейными выражениями от индексной переменной любого внешнего цикла (в этом случае нельзя гарантировать точность теста). На выходе получается список операторов, порождающих зависимость по данным.

В интерфейс системы встроен колоризированный текстовый редактор. Входная программа в результате лексического анализа разбивается на множество лексем, каждому типу которых соответствует свой цвет. Текст входной программы переводится в дерево внутреннего представления, в результате обхода которого происходит поиск операторов, “подозрительных” на зависимость. Далее происходит разбор индексных выражений и построение уравнения зависимости. Затем исполняются тест Шостака, Н.О.Д.-тест и тест Банержи и сравниваются результаты.

В дальнейшем данный тест может быть обобщен, что уменьшит начальные ограничения, и использован при написании компилятора в качестве блока анализа зависимости по данным.

СПИСОК ЛИТЕРАТУРЫ

1. **Shostak R.** Deciding linear inequalities by computing loop residues // J.ACM. — 1981. — Vol.28, №4. — P. 769–779.
2. **Евстигнеев В.А.** Основы параллельной обработки. Анализ программных зависимостей. — Новосибирск, 1996.

3. **Burke M., Cytron R.** Interprocedural dependence analysis and parallelization.: Tech. Rep. / IBM. — № 11794. — New-York, 1986.
4. **Banerjee U.** An introduction to formal theory of dependence analysis // J. Supercomputing. — 1988. — Vol. 2. — P. 133–149.
5. **Pugh W., Wonnacott D.** Static analysis of upper and lower bounds on dependences and parrallelism // J.ACM. — 1994. — Vol.16, № 4. — P. 248–278.
6. **Евстигнеев В.А.** Применение теории графов в программировании. — М.: Наука, 1985.
7. **Read R.C., Tarjan R.E.** Bounds on backtrack algorithms for listing cycles, paths, and spanning trees // ERL Memo M 433, Electronics Research Lab., Univ. Of California, Berkeley, Calif., 1973.
8. **Страуструп Б.** Язык программирования Си++. — М.: Радио и связь, 1991.
9. **Касьянов В.Н. Поттосин И.В.** Методы построения трансляторов. — Новосибирск: Наука, 1986.
10. **Girkar M., Polychronopoulos C.** Compiling issues for supercomputers: Rep. / CSRD, — №766. — Illinois, 1988.