

# Индексный анализ зависимостей по данным

А.Ю. Дроздов, Р.М. Корнев, А.С. Боханко

**Аннотация.** Работа посвящена компиляторной технологии, называемой индексным анализом, которая позволяет предоставить информацию о существовании зависимости между операциями, обращающимися к памяти и находящимися в цикле. Такая информация используется многими оптимизациями, особенно цикловыми. В работе для этой системы применяется алгоритм индексного анализа на основе системы линейных неравенств. Для формирования этой системы требуется выполнить подготовительные действия по поиску гнезд циклов и индуктивных переменных, нахождению инвариантов цикла, делинеаризации и непосредственному построению системы неравенств. Рассматриваются методика выполнения этих действий, временные параметры исполнения алгоритма, полученные результаты, а также некоторые особые случаи, не приведенные в [1].

## 1. Основные определения

**Определение 1.** Назовем две операции  $S$  и  $T$  в цикле *зависимыми*, если:

- $S$  и  $T$  обращаются к общему участку памяти  $M$ ;
- $S$  выполняется раньше  $T$  в цикле;
- между  $S$  и  $T$  нет других обращений к участку памяти.

**Определение 2.** Зависимость двух операций  $S$  и  $T$  может иметь разные направления, а именно:

- направление «больше» означает, что операция  $S$ , обращается к общему участку памяти  $M$  на некоторое количество итераций позже, чем операция  $T$ ;
- направление «меньше» означает, что операция  $S$ , обращается к общему участку памяти  $M$  на некоторое количество итераций раньше, чем операция  $T$ ;
- направление «равно» означает, что операции  $S$  и  $T$  обращаются к общему участку памяти  $M$  на одной итерации цикла.

Более подробное описание зависимостей, направлений зависимостей приведено в [1].

**Определение 3.** Под равенством  $S(i) = T(j)$  понимается, что данные операции обращаются к общему участку памяти.

**Определение 4.** Назовем ps-формой полином вида  $c_0 + c_1x_{11}x_{12}\dots x_{1k_1} + \dots + c_nx_{n1}x_{n2}\dots x_{nk_n}$ ,

где  $c_0, c_1, \dots, c_n$  - некоторые константы,  $x_{ij}$  - переменные.

Структура ps-формы приведена на рисунке.

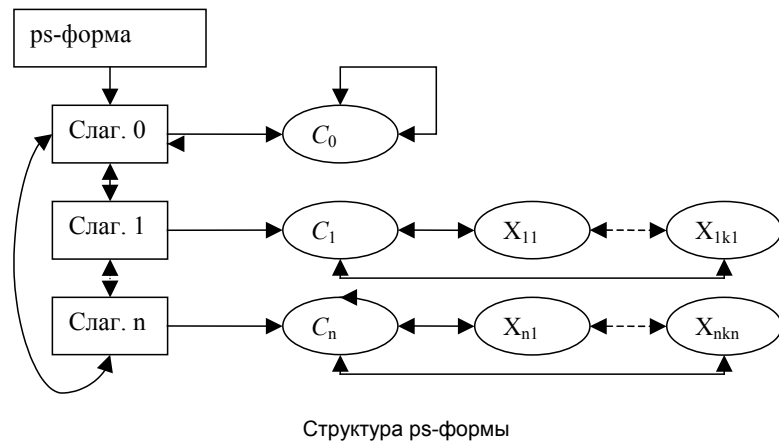
**Определение 5.** Def-Use графом называется ориентированный граф, узлами которого являются операции и ф-узлы; ф-узел, отнесенный к некоторой переменной, – это псевдооперация, выбирающая среди множества значений переменной нужное. Дуги Def-Use графа соответствуют потоку данных между операциями записей переменной и чтения.

## 2. Поиск гнезд циклов, для которых возможен индексный анализ

Прежде всего, напомним, что гнездом называется путь в дереве циклов [2] от корня до одного из листьев.

В работе [1] налагается множество ограничений на циклы и гнезда циклов, для которых возможно применение анализа; сумма таких ограничений сводится к понятию «совершенного цикла». Например,

в совершенном цикле должен быть только один выход, местоположение которого обязано совпадать с обратной дугой, и только одна индуктивная переменная, причем обязательно базовая (то есть имеющая верхнюю границу). Рассматриваемый в статье подход снимает практически все эти ограничения, многократно расширяя область применения анализа.



## 3. Поиск индуктивных переменных

Принцип выявления индуктивных (или «индексных», согласно [1]) переменных основан на использовании Def-Use графа [2]. Индуктивная переменная обладает следующими свойствами:

- операции и φ-узлы, работающие с индуктивной переменной  $V$  в цикле  $L$ , образуют в Def-Use графе сильно-связную компоненту  $C$ ;
- в компоненту  $C$  входит φ-узел, стоящий в голове цикла  $L$ ;
- можно доказать, что на каждой итерации цикла  $L$  значение переменной  $V$  изменяется на одну и ту же величину.

Верно и обратное: если для некоторой переменной все три перечисленных свойства соблюдаются, то такая переменная является индуктивной.

На основе последнего утверждения практически определяется и само понятие индуктивной переменной, и алгоритм поиска таких переменных:

1. В Def-Use графе находится сильно-связная компонента  $C$ .
2. Вычисляется к какому циклу  $L$  принадлежит компонента  $C$ .
3. Проверяется, что в компоненту  $C$  входит φ-узел  $P$ , стоящий в голове цикла  $L$ . Переменная  $V$ , определяемая φ-узлом  $P$ , является кандидатом на роль индуктивной переменной цикла  $L$ .
4. Проверяется, что все операции цикла  $L$ , изменяющие значение переменной  $V$ , принадлежат компоненте  $C$ .
5. Доказывается, что на любой итерации цикла  $L$  значение переменной  $V$  изменяется на одну и ту же величину, инвариантную относительно итераций цикла  $L$ .

Если все пять шагов алгоритма успешно пройдены, то  $V$  является индуктивной переменной цикла  $L$ .

После того, как индуктивная переменная найдена, необходимо найти ее нижнюю и верхнюю границы и шаг. Значения этих трех величин можно представить в виде ps-форм.

Нижняя граница – это ps-форма операции, определяющей значение переменной  $V$  перед входом в цикл  $L$ . Если такую операцию найти нельзя (например, в том случае, когда нет одной операции, доминирующей над входом в цикл), то нижняя граница неизвестна.

Для нахождения ps-формы шага следует вычислить ps-форму операции приращения переменной  $V$  (это последняя операция компоненты  $C$ ), а затем вычесть из нее  $V$  (так как операция прира-

щения входит в сильно - связную компоненту с  $\phi$ -узлом, определяющим  $V$ , то переменная  $V$  в ps-форме так или иначе присутствует). Полученная в результате ps-форма и будет шагом.

Верхняя граница – это ps-форма операции, которая определяет значение, проверяемое при выходе из цикла. Если выходов несколько, то можно вычислить лишь максимальную верхнюю границу и только для тех переменных, нижняя граница и шаг которых известны. Индуктивная переменная с известными нижней и верхней границами называется базовой.

Проиллюстрируем сказанное выше примером:

```
j = 0;
for ( i = 0; i < N * M + 3 * L; i += 3 )
{
    ...
    j++;
}
```

В данном цикле две индуктивные переменные –  $i$  и  $j$ ; при этом переменная  $j$  является базовой. Нижняя граница переменной  $j$  равна 0; верхняя граница равна значению выражения  $N * M + 3 * L$ .

Рассмотрим подробнее, как выглядит ps-форма верхней границы  $1 * N * M + 3 * L$ . В ней обозначены два слагаемых ( $1 * N * M$  и  $3 * L$ ), состоящие из трех и двух сомножителей соответственно. Сомножители  $N$ ,  $M$  и  $L$  – это контейнеры, внутри которых хранятся ссылки на узлы Def-Use графа, определяющие значения соответствующих переменных непосредственно перед циклом.

#### 4. Поиск инвариантов гнезда циклов

Перед запуском анализа существования зависимых операций в цикловом регионе необходимо найти все инвариантные операции и объекты циклов, находящихся в гнезде. Определение инвариантной операции дается в [2]. Инвариантной называется операция результат выполнения которой одинаков для всех итераций цикла. (Таким образом, об инвариантной операции можно говорить только в контексте некоторого цикла). Операция может оказаться инвариантной относительно одного цикла, но не инвариантной относительно другого цикла данного гнезда. Алгоритм принятия решения об объявлении операции инвариантной не привязан к архитектуре микропроцессора. Объект объявляется инвариантным, если его формирует инвариантная операция. Разделение объектов и операций на инвариантные и неинвариантные используется при подготовке матричного представления исследуемых операций. После нахождения инвариантов рассматриваемого цикла строятся ps-формы для операций чтения и записи в память, принадлежащих циклу.

#### 5. Сохранение информации об измерениях

Сохранение и использование информации об измерениях многомерных массивов позволяет уточнить анализ.

Приведем пример: две операции программы, написанной на языке C, обращаются к элементам одного двумерного массива:

```
int a[100][100];
...
a[i * 2][j] = foo;
a[i * 2 + 1][j] = bar;
```

Предположим, что шаг приращения индуктивных переменных  $i$  и  $j$  известен и равен 1, а границы этих переменных неизвестны. В таком случае только отдельный анализ для старшего измерения и утверждение о том, что значение индекса младшего измерения не может превышать размер этого измерения (такое утверждение является правилом хорошего тона в одних языках, например, C, и требованием стандарта в других - Фортране, Паскале), позволяет доказать независимость операций.

Пусть в промежуточном представлении программы выражения, вычисляющие адрес, представлены следующим образом:

$$\text{Address1} = a + (i * 2) * 100 + j$$

$$\text{Address2} = a + (i * 2 + 1) * 100 + j$$

Это представление уже не позволяет выделить выражения, вычисляющие индексы отдельных измерений, но локальные оптимизации могут сделать ситуацию еще хуже:

$$\text{bar} = a + 200 * i + j$$

$$\text{Address1} = \text{bar}$$

$$\text{Address2} = \text{bar} + 100$$

Теперь выделить выражения для отдельных измерений и вовсе невозможно.

Предлагается на этапе компиляции ввести временные переменные для хранения размерностей отдельных измерений многомерных массивов. В таком случае выражения, вычисляющие адрес, будут иметь вид, подобный следующему:

$$\text{Address1} = a + (i * 2) * a\_dim1\_size + j$$

$$\text{Address2} = a + (i * 2 + 1) * a\_dim1\_size + j$$

Выделение выражения индекса старшего измерения стало тривиальным: в него входят все слагаемые с множителем  $a\_dim1\_size$ . В младшее измерение входят слагаемые без множителей – временных переменных.

Разумеется, после заведения временных переменных поле действия локальных оптимизаций сужается – ведь в тех местах, где раньше находились константы, теперь находятся переменные с неизвестными значениями. Поэтому рано или поздно значения временных переменных придется раскрыть. Но можно отложить момент раскрытия на поздние этапы работы компилятора, когда вся необходимая от анализа зависимостей информация уже будет получена.

Известны также другие подходы к получению информации об измерениях [2]. В их основе лежит алгоритмическая декомпозиция выражений, вычисляющих адрес. Очевидная слабость таких подходов – ограниченность их применения и не всегда оптимальные результаты.

## 6. Подготовка данных гнезда циклов

Подготовка данных начинается с подсчета числа циклов и индуктивных переменных в гнезде циклов. Для этого берется внутренний цикл гнезда, не включающий вложенных циклов, и начинается последовательный перебор охватывающих циклов вплоть до охватывающего цикла всего гнезда (включительно) или до цикла, в котором нет индуктивных переменных. Для каждого рассматриваемого цикла уже известно число индуктивных переменных, так что надо прибавить его к общему числу индуктивных переменных гнезда, которое в начале перебора равно нулю.

На следующем этапе заполняется (перебором от внутреннего до охватывающего цикла гнезда) массив номеров индуктивных переменных и массив их шагов. Алгоритм заполнения состоит в следующем.

1. Для каждой переменной рассматриваемого цикла определяется, следует ли вносить ее в массив индуктивных переменных гнезда. Если шаг переменной задается константой (включая нулевое значение), то переменная не вносится в массив.

2. Если было принято положительное решение, то шаг переменной вносится в массив шагов переменных гнезда. Если рассматриваемая – базовая, то в массив номеров переменных гнезда она вносится под специальным номером (этот номер одинаков для всех базовых переменных и достаточно велик, чтобы быть больше числа индуктивных переменных в гнезде), в противном случае в массив номеров вносится номер базовой переменной, к которой может быть сведена рассматриваемая (а именно, последняя рассматриваемая базовая переменная).

После подготовки массива номеров переменных и массива их шагов надо сформировать систему неравенств, определяющую верхние и нижние границы переменных. Аналогичным образом, перебором от внутреннего до охватывающего цикла гнезда, для каждой индуктивной переменной

определяется, 1. есть ли у нее верхняя и нижняя границы, 2. подсчитывается число неравенств (неравенства в виде ps-форм получаются на этапе поиска индуктивных переменных). Существование границы подразумевает существование определяющего ее неравенства. Затем, проведя последовательный просмотр индуктивных переменных каждого цикла, алгоритм выделяет из ps-форм верхних и нижних границ (если они существуют) соответствующие им неравенства. Матрица границ устроена следующим образом: по строкам записаны неравенства, по столбцам переменные. Постоянная часть ps-формы записывается в вектор свободных членов.

Итогом подготовки данных гнезда являются массивы шагов переменных, номеров переменных, матрица границ и вектор, соответствующий свободным членам неравенств, задающих границы.

## **7. Подготовка данных для анализа существования зависимости двух операций обращения к массиву в цикле**

На данном этапе входными данными являются операции, для которых будет производиться анализ, и цикл, которому принадлежат данные операции и соответствующее гнездо циклов с уже подготовленными данными.

Для каждого измерения массива, к которому обращаются исследуемые операции, выполняется следующий алгоритм.

1. Из ps-формы операции выделяется часть, соответствующую текущему измерению.
2. Если это последнее измерение, то выделяется смещение из адресных констант, которое затем используется в качестве ps-формы данного измерения.
3. ps-формы данного измерения разделяются относительно данного цикла на инвариантную и неинвариантную части. Если разность инвариантных частей ps-форм данного измерения представляется константой, то дальнейшее разделение останавливается. Если это не так, то производится разделение относительно самого охватывающего цикла из гнезда.
4. Из инвариантной части для первой операции вычитается инвариантная часть для второй операции.
5. Если разность – не константа или одна из ps-форм не пригодна для анализа (есть нелинейность, множитель – не индуктивная переменная, шаг индуктивной переменной не является константой) то проводить анализ нельзя, выдается ответ о наличии зависимости. Конец работы алгоритма.
6. Разность неинвариантных частей добавляется к неинвариантной части ps-формы второй операции.
7. Заполняется матрица, соответствующая системе неравенств. Константная часть ps-формы записывается в вектор свободных членов для данной операции. Для каждого множителя, не равного константе, находится соответствующая ему индуктивная переменная, и к значению, записанному в строке с номером, равным рассматриваемому измерению, и столбце, с номером, равным номеру индуктивной переменной, прибавляется коэффициент при данном множителе.

## **8. Интерфейс драйвера алгоритма анализа зависимостей**

После того, как основные данные подготовлены, возможен вызов драйвера алгоритма анализа зависимостей. При вызове может решаться одна из следующих задач:

1. Поиск зависимости в заданном направлении.
2. Поиск зависимостей поочередно в трех направлениях - «больше», «меньше» или «равно» - для текущей и всех зависимых от нее переменных. Если в каком-то из направлений зависимость присутствует, оно фиксируется для текущей переменной, а поиск продолжается рекурсивно для следующих переменных.
3. Поиск зависимости ведется для всех переменных во всех направлениях.

Кроме поставленной задачи в драйвер алгоритма передаются данные, общие для разных вызовов. В их состав входят: система линейных неравенств, задающая верхние и нижние границы переменных (подготовка описана в разделе 6), система линейных уравнений, описывающая операции, анализируемые на зависимость (подготовка описана в разделе 7). Кроме этого, в драйвер передается вектор номеров переменных (описанный в разделе 6) и вектор шагов, заданных для каждой переменной. В дальнейшем эта информация будет использована при приведении шагов переменных к единичным.

## 9. Подготовка данных к вызову алгоритма анализа зависимостей

После того, как были подготовлена информация о гнезде циклов (матрица границ, вектор номеров переменных) и информация об анализируемых операциях, необходимо сделать несколько финальных преобразований. Они необходимы для правильной работы алгоритма, анализирующего операции на предмет зависимости. Их описание приводится ниже.

Пусть система линейных неравенств, определяющих границы, задается матрицей  $Q$  (с размером  $k \times m$ , где  $m$  - число переменных в гнезде циклов,  $k$  - количество неравенств) и вектором свободных членов  $q_0$ . Система линейных уравнений, определяющая доступ к памяти первой операции, задается матрицей  $A$  и вектором  $a_0$ , второй операции – матрицей  $B$  и вектором  $b_0$ .

Шаг 1. Разделение матрицы границ на матрицы верхних и нижних границ.

Необходимо разделить систему на неравенства, отвечающие за нижнюю границу (матрица  $out\_p$  и вектор  $out\_p_0$ ), и за верхнюю границу (матрица  $out\_q$  и вектор  $out\_q_0$ ). Для этого используется следующий алгоритм.

1. В каждой строке матрицы  $Q$ , начиная с верхней, ищется ненулевой элемент.

Если такого не нашлось, а соответствующий свободный член больше или равен нулю, то выполняется переход к следующей строке, иначе - переход к следующему шагу алгоритма.

2. Если найденный элемент больше нуля,

то выполняется копирование строки с номером, равным номеру столбца найденного ненулевого элемента, в матрицу  $out\_q$ , а элемента из  $q_0$  в вектор  $out\_q_0$ ,

иначе строка копируется в матрицу  $out\_p$ , а элемент из  $q_0$  - в вектор  $out\_p_0$ .

Шаг 2. Преобразование отрицательного шага переменных к положительному.

После выделения матриц, необходимо умножить на  $-1$  строки в матрицах  $out\_q$ ,  $out\_p$  и элементы векторов  $out\_q_0$ ,  $out\_p_0$ , соответствующие переменным с отрицательным шагом. После этого отрицательные шаги изменяются на положительные, для чего матрицы  $out\_p$ ,  $out\_q$ ,  $A, B$  изменяются следующим образом:

- в матрицах  $A, B, out\_p, out\_q$  столбцы, соответствующие переменным с отрицательным шагом, умножаются на  $-1$ .

В векторах  $out\_p_0, out\_q_0$  изменений не производится.

Шаг 3. Приведение переменных к общему виду.

Следующим шагом является приведение границ переменных к общему виду:

$$0 \leq x_i \leq a_i.$$

Пусть  $i$  - номер рассматриваемой переменной. Тогда алгоритм приведения переменных к общему виду выглядит следующим образом:

$$- C_p = -out\_p[i][i], C_{\min} = out\_p_0[i].$$

Если  $C_p = 0$ , то выполняется переход к следующей переменной.

Сохраняется строка  $out\_p$  с номером  $i$  в векторе  $min$ .

- Пусть  $j = 0, \dots, m-1$ .

Для каждого  $j$ ,

если  $out\_q[j][j] \neq 0$ , то строка матрицы  $out\_q$  и элемент  $out\_q_0$  с номером  $j$  умножаются на  $C_p$ ,

аналогично, если  $out\_p[j][j] \neq 0$ , то строка матрицы  $out\_p$  и вектор  $out\_p_0$ , умножаются на  $C_p$ .

Затем, если  $out\_q[j][j] \neq 0$ ,  $out\_p[j][j] \neq 0$ , то каждый элемент строки матриц  $out\_p$  и  $out\_q$  умножается на  $1 + \min[k]$ , где  $k$  - номер столбца элемента.

К векторам  $out\_p_0[j]$ ,  $out\_q_0[j]$  прибавляются, соответственно,  $out\_p[j][i] \cdot C_{min}$ ,  $out\_q[j][i] \cdot C_{min}$ .

- Строки с номером  $j$  матриц  $A$  и  $B$ ,  $a_0[j]$ ,  $b_0[j]$  умножаются на  $C_p$ .

Затем каждый элемент  $j$ -ой строки умножается на  $1 - \min[k]$ , где  $k$  - номер столбца элемента.

После этого из  $a_0[j]$  и  $b_0[j]$  вычитается  $A[j][i] \cdot C_{min}$ ,  $B[j][i] \cdot C_{min}$ .

Переход к следующей переменной.

Шаг 4. Приведение шагов переменных к единичному.

Алгоритм, описанный в [1], можно применять только в случае единичного шага переменных. После приведения переменных к общему виду можно заменить приращение на шаг умножением на шаг, например:

$i=0, \text{step}=3, i=0,3,6, \dots$

$i=0, \text{step}=1, 3*i=0,3,6, \dots$

Это преобразование возможно только в том случае, если для данной переменной определена нижняя граница и ее коэффициенты в матрицах выражений совпадают.

Шаг 5. Удаление сводимых индуктивных переменных.

Следующий важный шаг – удаление индуктивных переменных, которые можно свести к базовым. Для этого в векторе номеров переменных ищется переменная, не помеченная специальным номером (не являющаяся базовой). Если для нее существуют и верхняя, и нижняя границы, то такую переменную можно удалить. Пусть  $i$  - номер удаляемой переменной,  $magic$  - номер переменной, к которой сводится удаляемая переменная. Алгоритм удаления переменной состоит в следующем.

- Изменение коэффициентов базовой переменной. В матрицах  $A$  и  $B$  к столбцу с номером  $magic$  прибавляется столбец с номером  $i$ . С матрицами  $out\_p$  и  $out\_q$  производится аналогичная операция (элемент, находящийся в  $i$ -ом столбце на  $i$ -ой строке не прибавляется).

- Изменение системы неравенств.

Пусть  $k = i + 1, \dots, m$ ,  $j = 1, \dots, m$ .

Выполняется  $out\_p[k-i][j] = out\_p[k][j]$  и аналогичное преобразование для матрицы  $out\_q$ .

Далее, выполняется  $out\_p[j-i][k] = out\_p[j][k]$  и аналогичное преобразование для  $out\_q$ .

- Приведение неравенств к новому количеству переменных ( $m-1$ ).

Пусть  $k = 2, \dots, m-1$ ,  $j = 1, \dots, m-1$ .

Выполняется  $out\_p[k-j][j] = out\_p[k][j]$  и аналогичное преобразование для  $out\_q$ .

- Изменение уравнений.

Пусть  $k = i + 1, \dots, m$ ,  $j = 1, \dots, n$ .

Выполняется  $A[k-1][j] = A[k][j]$  и аналогичное преобразование для матрицы  $B$ .

- Приведение уравнений к новому количеству переменных ( $m-1$ ).

Пусть  $k = 1, \dots, m$ ,  $j = 1, \dots, n$ .

Выполняется  $A[k-j][j] = A[k][j]$  и аналогичное преобразование для  $B$ .

- Изменение числа переменных.

Уменьшается на 1 число индуктивных переменных.

Корректируется вектор переменных.

Шаг 6. Добавление неравенств.

Последним шагом перед работой алгоритма анализа зависимостей является добавление неравенств, задающих направление поиска зависимостей для выбранной переменной. Пусть  $i(t)$ ,  $j(t)$  - вектора значений индуктивных переменных, при которых операции обращаются к общему участку памяти,  $k$  - номер переменной, для которой ставится задача. Это означает, что для направления «меньше» будет добавлено неравенство:

$$i[k] - j[k] > 0,$$

а для направления «больше»:

$$i[k] - j[k] < 0.$$

## 10. Особенности алгоритма анализа зависимостей

К особенностям алгоритма можно отнести используемый для решений систем линейных неравенств целочисленный симплекс-метод (ЦСМ). Алгоритм используемого ЦСМ приведен ниже (ССН – список систем неравенств).

0. Начало.

1. Создается пустой ССН, и в него помещается исходная система неравенств.

Если линейная форма на заданной системе неравенств не ограничена, то выполняется останов.

2. Если система Parent, находящаяся в начале списка, имеет нецелые решения, то строятся новые системы Child\_1 и Child\_2, каждая из которых содержит все неравенства системы Parent, а также одно дополнительное неравенство, осуществляющее отсечения сверху (система Child\_1) и снизу (система Child\_2) по переменной, соответствующей первому нецелому компоненту решения системы Parent.

Совокупность систем Child\_1 и Child\_2 имеет целочисленные решения тогда и только тогда, когда их имеет система Parent.

3. Система Parent удаляется из списка.

4. Новые системы добавляются в список (если система не имеет решений, то вместо добавления она уничтожается). Вставка осуществляется таким образом, чтобы системы неравенств в списке были упорядочены по убыванию соответствующих оптимальных значений линейной формы.

5. Если на данном шаге список пуст, то все возможные варианты построения новых систем исчерпаны, следовательно, у исходной системы неравенств нет целочисленных решений. Конец.

6. Если новая система Parent имеет целочисленное решение, то:

а) значение линейной формы, соответствующее этому решению, является максимальным для системы Parent (т.е. максимальным на части исходного многогранника, определяемой системой Parent);

б) ввиду того, что Parent является первой системой в списке, и список упорядочен по убыванию экстремальных значений линейной формы, соответствующих его звеньям, то

найденное значение максимально на всем исходном многограннике. Следовательно, целочисленное решение системы Parent и есть решение исходной задачи. Конец.

#### 7. Переход к шагу 1.

Кроме целочисленного симплекс-метода, реализованы рациональный симплекс-метод [4] и метод Фурье [1].

### 11. Минимальное расстояние зависимости

Пусть даны два выражения  $S(i)$  и  $T(j)$ ,  $S(i) = T(j)$ . Назовем вектором расстояний векторную разность  $i(t) - j(t)$  (здесь  $i(t), j(t)$  - вектора значений индуктивных переменных, при которых операции обращаются к общему участку памяти). Это означает, что для каждой переменной в векторе расстояний записано число, показывающее через сколько итераций будет пересечение по памяти. Для каждой переменной можно найти минимальное число итераций. Поиск минимального расстояния в основном нужен в случае единственной индуктивной переменной. Все методы, используемые для решения системы линейных неравенств - минимаксные, поэтому минимальный вектор (если он существует), получается как дополнительный результат. Значение минимального расстояния зависимости используется в важных цикловых оптимизациях, например тех, что описаны [5].

### 12. Интерпретация и использование результатов анализа в целях оптимизации

Результатом анализа являются установление: наличия зависимости, наличия зависимости в данном направлении, существования вектора расстояний, независимости операций. Существование минимального вектора расстояний используется при вычислении размеров рекуррентностей в наложенных циклах, а также позволяет переставлять операции местами, если известно что на одной итерации зависимости между ними нет.

### 13. Экспериментальные результаты

Ниже приведены экспериментальные результаты запуска алгоритма анализа зависимостей с разными алгоритмами решения системы линейных неравенств. Все алгоритмы запускались на тесте, приведенном ниже.

```
#include <stdio.h>

int array[15][15][15][15][15][15][15];

void main(void)
{
    int i1,i2,i3,i4,i5, j2, j3;
    int array1[15][15][15][15][15][15][15];

    for ( i1 = 0; i1 < 10; i1++ )
    {
        for ( i2 = 2 * i1 + 1; i2 < i1 - 6; i2++ )
        {
            for ( i3 = i1 + 3 * i2 - 1, j3 = i1; i3 < 2 * i1 - i2 ; i3++, j3+=3 )
            {
```



систем линейных неравенств, особенно метод Фурье. В сравнении с существующими алгоритмами (Power Test [6], Omega Test [7]) используемый симплекс-метод выигрывает по времени работы. Power Test является комбинацией метода Фурье с расширенным тестом на наибольший общий делитель, а Omega Test - расширение метода Фурье. Оба алгоритма широко применимы и обеспечивают высокую точность проверки на зависимость, но в худшем случае имеют экспоненциальную сложность. Симплекс-метод также имеет экспоненциальную сложность в худшем случае, но на практике такой случай недостижим. На реальных же задачах симплекс-метод выигрывает по скорости работы у метода Фурье. Использование целочисленного симплекс-метода позволяет добиться хороших результатов по точности, не сильно ухудшая время вычислений. Похожий метод, например, используется в работе [8] (I-test).

## Литература

1. Utpal Banerjee, "Loop Transformations for Restructuring Compilers", Vols. 1-3, Kluwer, Boston, 1993, 1994, 1997.
2. Steven S. Muchnick, "Advanced Compiler Design and Implementation", Morgan Kaufmann, San Francisco, 1997, Section 8.10.
3. Vadim Maslov, "Delinearisation: an Efficient Way to Break Multiloop Dependence Equations", Proceedings of PLDI'92, ACM, 1992.
4. Г.П. Кюнц, В. Крелле, "Нелинейное программирование", Москва, Советское Радио, 1965.
5. P.P. Tiрумалай, M. Lee, M. S. Schlansker, "Parallelization of loops with exits on pipelined architectures", Supercomputing, pp 200-212, November 1990.
6. M. Wolfe, C.W. Tseng, "The Power test for data dependence", IEEE Transaction on Parallel and Distributed Systems, Vol. 3, No. 5, pp. 591-601, September 1992.
7. W. Pugh, "A practical algorithm for exact array dependence analysis", Communication of the ACM, 35(8): 102-114, August 1992.
8. K. Psarris, K. Kyriakopoulos, "Data Dependence Testing in Practice", IEEE International Conference on Parallel Architectures and Compiler Techniques, October 12-16, 1999, California.
9. D.E. Maydan, J.L. Hennesy, M.S. Lam "Efficient and Exact Data Dependence Analysis", Proceedings of the ACM SIGPLAN' 91 Conference on Programming Language Design and Implementation.

**Дроздов Александр Юльевич.** Родился в 1966 г. Окончил МГУ им. М.В. Ломоносова в 1988 г. Автор 11 научных работ. Область научных интересов – технологии оптимизирующей компиляции и программной инженерии. И.о. начальника отдела в Институте Микропроцессорных Вычислительных Систем РАН.

**Корнев Роман Михайлович.** Родился в 1980 г. Окончил МГУ им. М.В. Ломоносова в 2003 г. Область научных интересов - оптимизирующая компиляция. Младший научный сотрудник ЗАО "МЦСТ".

**Боханко Андрей Сергеевич.** Родился в 1979 г. Окончил Московский технический университет связи и информатики в 2001 г. Область научных интересов – оптимизирующая компиляция, программная инженерия. Аспирант Института Микропроцессорных Вычислительных Систем РАН.