

Индексный анализ зависимостей по данным

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

ИМВС РАН, г. Москва

[sasha|corneff|ruff}@mcst.ru](mailto:{sasha|corneff|ruff}@mcst.ru)

Введение

В статье описывается компиляторная технология, называемая индексным анализом, позволяющая предоставить информацию о зависимости (независимости) операций, обращающихся к памяти и находящихся в цикле. Данная информация используется многими оптимизациями, особенно цикловыми. В [1] описан алгоритм, позволяющий, на основе системы линейных неравенств, дать ответ, есть зависимость или нет. Однако для получения системы линейных неравенств необходимо сделать несколько подготовительных шагов, заключающихся в поиске гнезд циклов, индуктивных переменных, нахождении инвариантов цикла, делинеаризации и, собственно, получение матричного представления на основе информации, полученной ранее. Ниже вкратце описаны вышеуказанные шаги, а также приведены результаты работы алгоритма по времени, интерпретация получаемых результатов и некоторые исключения, не описанные в [1]. Данные исключения описаны в разделе 10.

1. Основные определения

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

1. S и T обращаются к общему участку памяти M .
2. S выполняется раньше T в цикле.
3. Между S и T нет других обращений к участку памяти.

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

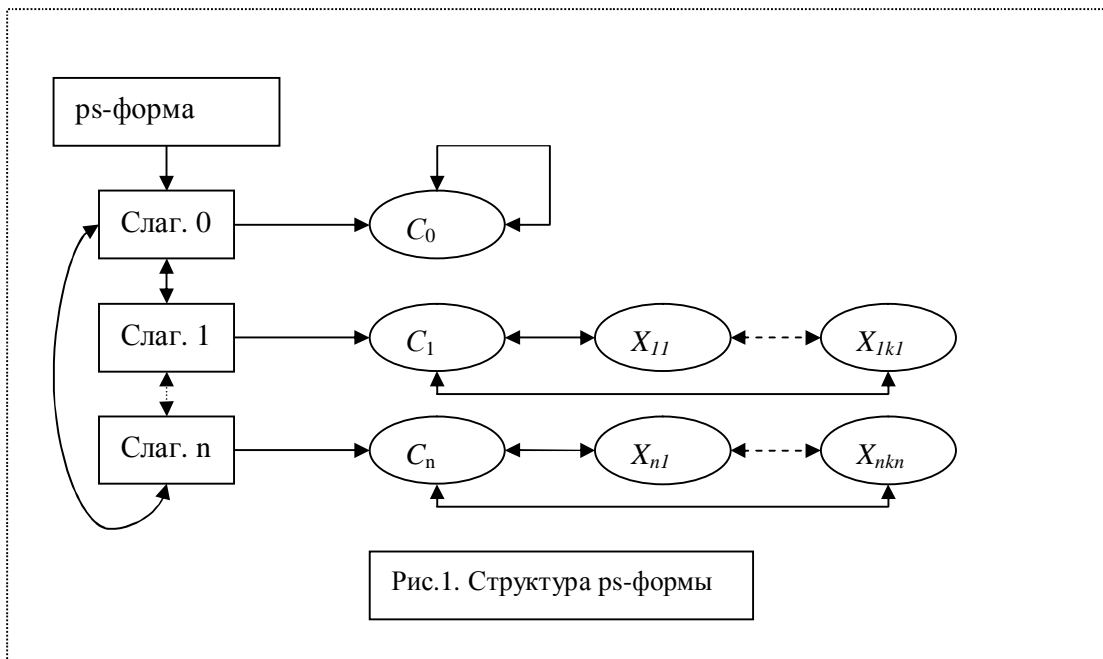
1. Направление «больше» означает, что операция S , обращается к общему участку памяти M на некоторое количество итераций позже, чем операция T .
2. Направление «меньше» означает, что операция S , обращается к общему участку памяти M на некоторое количество итераций раньше, чем операция T .
3. Направление «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.

Рис. 1. иллюстрирует внутреннее строение ps-формы в полном соответствии с данным выше определением.



Определение 5. Def-Use графом называется ориентированный граф, узлами которого являются операции и ф-узлы; дуги Def-Use графа выражают поток данных между соответствующими узлами. ф-узел для некоторой переменной – это псевдооперация, выбирающая среди множества значений переменной нужное.

2. Поиск гнезд циклов, для которых возможен анализ

Первая задача, которую надо решить – поиск гнезд циклов, для которых возможно проведение индексного анализа.

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

Как было сказано выше, наш подход основан на работе [1]. Но в упомянутой работе налагается множество ограничений на циклы и гнезда циклов, для которых возможно применение анализа; сумма таких ограничений образует понятие «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). Мы снимаем практически все эти ограничения, многократно расширяя, таким образом, область применения анализа.

Список остающихся ограничений крайне немногочислен и практически не уменьшаем: циклы должны быть сводимыми и иметь хотя бы одну индуктивную переменную.

3. Поиск индуктивных переменных

Как уже было отмечено, выявление индуктивных переменных (иногда используется термин индексные переменные, [1]) жизненно необходимо для полноценного анализа зависимостей.

Наш подход основан на использовании Def-Use графа [2]. А именно, индуктивная переменная обладает следующими свойствами:

1. Операции и ϕ -узлы, работающие с индуктивной переменной V в цикле L , образуют в Def-Use графе сильно-связную компоненту C ,
2. В компоненту C входит ϕ -узел, стоящий в голове цикла L ,
3. 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.

Сделав последнее утверждение, мы практически определили и само понятие индуктивной переменной, и алгоритм поиска таких переменных:

1. Находим в Def-Use графе сильно-связную компоненту C ,
2. Вычисляем, какому циклу L принадлежит компонента C ,
3. Проверяем, 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. Проверяем, что все операции цикла L , изменяющие значение переменной V , принадлежат компоненте C ,
5. Доказываем, 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, значит V является индуктивной переменной цикла L .

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

Нижняя граница – это rs-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), значит, нижняя граница неизвестна.

Для нахождения rs-формы шага следует вычислить rs-форму операции приращения переменной V (это последняя операция компоненты C), а затем вычесть из нее V (так как операция приращения входит в сильно-связную компоненту с ϕ -узлом, определяющим V , то переменная V в rs-форме так или иначе присутствует). Полученная в результате rs-форма и будет шагом.

Верхняя граница – это rs-форма операции, определяющей значение, проверяемое при выходе из цикла. Если выходов несколько, то можно вычислить только 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 rs-форма верхней границы:

```
1 * N * M + 3 * L
```

В данной rs-форме два слагаемых ($1 * N * M$ и $3 * L$), состоящих 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 зависимостей операций в цикловом регионе необходимо найти все инвариантные операции и объекты циклов, находящиеся в гнезде. Определение инвариантной операции дается в [2]. Операция может оказаться инвариантной относительно одного цикла, но не инвариантной относительно другого цикла из 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 не инвариантные 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 анализу зависимостей давать более точные ответы.

Приведем пример. Допустим, даны две операции программы, написанной на языке C, обращающиеся 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 нам 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 (Фортран, Паскаль)), позволяет нам доказать независимость операций.

В промежуточном представлении программы выражения, вычисляющие адрес, очевидно, представлены следующим образом:

```
Address1 = a + (i * 2) * 100 + j  
Address2 = a + (i * 2 + 1) * 100 + j
```

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

```
bar = a + 200 * i + j  
Address1 = bar  
Address2 = bar + 100
```

Теперь выделить выражения для отдельных измерений стало и вовсе невозможно.

Мы предлагаем завести на этапе компиляции специальные временные переменные для хранения размерностей отдельных измерений многомерных массивов. В таком случае выражения, вычисляющие адрес, будут иметь подобный вид:

$$\text{Address1} = a + (i * 2) * a_dim1_size + j$$
$$\text{Address2} = a + (i * 2 + 1) * a_dim1_size + j$$

Теперь выделение выражения индекса старшего измерения тривиально: в него входят все слагаемые с сомножителем `a_dim1_size`. В младшее измерение входят слагаемые без сомножителей – временных переменных.

Разумеется, после заведения временных переменных поле действия локальных оптимизаций сужается – ведь в тех местах, где раньше стояли константы, теперь находятся переменные с неизвестными значениями. Поэтому рано или поздно значения временных переменных придется раскрыть. Но мы можем отложить момент раскрытия на поздние этапы работы оптимизирующего 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 самый вложенный цикл и начинается подъем вверх по циклам до тех пор, пока не дойдем до самого охватывающего цикла 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 (перед началом работы алгоритма общее число индуктивных переменных равно нулю).

На следующем этапе заполняется массив номеров индуктивных переменных и массив их шагов, начиная с самого вложенного цикла гнезда до самого верхнего. Алгоритм заполнения состоит в следующем.

1. Для каждой переменной рассматриваемого цикла определяется, вносить ее в массив индуктивных переменных гнезда или нет. Если у рассматриваемой переменной не константный шаг, или константа равна нулю, такая переменная не вносится в массив.
2. Если было принято решение о занесении в массив переменных, тогда шаг переменной заносится в массив шагов переменных гнезда. Если рассматриваемая

переменная базовая, то в массив номеров переменных гнезда она заносится под специальным номером (данный номер одинаков для всех базовых переменных и достаточно велик, чтобы быть больше числа индуктивных переменных в гнезде), в противном случае в массив номеров заносится 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 заполнить матрицу неравенств, определяющую верхние и нижние границы переменных. Аналогичным образом, от самого вложенного цикла к самому охватывающему для каждой индуктивной переменной определяется, есть ли у нее верхняя, нижняя границы и подсчитывается число неравенств (неравенства в виде ps-форм получаются на этапе поиска индуктивных переменных). Существование границы подразумевает существование неравенства, определяющего ее. Затем в цикле по индуктивным переменным каждого цикла из ps-форм верхних и нижних границ (если они существуют) выделяются соответствующие им неравенства. Матрица границ устроена так: по строкам записаны неравенства, по столбцам переменные. Константная 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 на зависимость двух операций обращения к массиву в цикле

На данном этапе входными данными являются операции, для которых будет производиться анализ, 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. Выделяем из ps-формы операции часть, соответствующую текущему измерению.
2. Если это последнее измерение, выделяем смещения из адресных констант и используем дальше его в качестве ps-формы данного измерения.
3. Разделяем ps-формы данного измерения на инвариантную и неинвариантную части относительно данного цикла. Если разность инвариантных частей ps-форм данного измерения для операций константа, останавливаемся на таком разделении. Если это не так, тогда производим деление на инвариантную и неинвариантную части относительно самого охватывающего цикла из гнезда.
4. Вычитаем из инвариантной части для первой операции инвариантную 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) – проводить анализ нельзя, выдается ответ, что зависимость есть. Конец работы алгоритма.
6. Разность неинвариантных частей добавляем к неинвариантной части ps-формы второй операции.
7. Заполняем матрицы. Константная часть ps-формы записывается в вектор свободных членов для данной операции. Для каждого не константного множителя ищем индуктивную переменную, соответствующую множителю и к значению, записанному в строке с номером, равным рассматриваемому измерению, и столбце, с номером, равным номеру индуктивной переменной, прибавляем коэффициент при данном множителе.

8. Вызов интерфейса драйвера алгоритма анализа зависимостей

После того, как основные данные подготовлены, возможен вызов драйвера алгоритма анализа зависимостей. При вызове может быть поставлена одна из нескольких задач, а именно:

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

Кроме поставленной задачи в драйвер алгоритма передаются данные, общие для разных вызовов. В следующем абзаце представлено описание этих данных.

В драйвер алгоритма передаются 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 переменных. В векторе шагов переменных для каждой индуктивной переменной указан ее шаг. В дальнейшем данная 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. Эти преобразования необходимы для правильной работы алгоритма, анализирующего операции на предмет зависимости. Описание данных преобразований приводится ниже.

Пусть система линейных неравенств, определяющих границы, задается матрицей Q (размера $k \times m$, 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, ищем ненулевой элемент. Если такого элемента не нашлось, а соответствующий свободный член больше или равен нулю – переходим к следующей строке. Если такой элемент найден, переходим к следующему шагу алгоритма.
2. Если найденный элемент больше нуля, копируем строку с номером, равным номеру столбца найденного нами ненулевого элемента, в матрицу out_q , элемент из q_0 в out_q_0 , иначе копируем в матрицу out_p , свободный член из q_0 в out_p_0 .

Шаг 2. Преобразование отрицательного шага переменных к положительному.

После выделения матриц, необходимо строки в матрицах out_q , out_p и элементы out_q_0 , out_p_0 , соответствующие переменным с отрицательным шагом, умножить на -1 . После данного преобразования меняем отрицательные шаги на положительные, для этого изменяем матрицы out_p , out_q , A , B 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 $C_p = 0$, переходим к следующей переменной. Сохраняем строку out_p с номером i в векторе \min .
- пусть $j = 0, \mathbf{K}, m-1$, тогда для каждого j , если $out_q[j][j] \neq 0$, умножаем строку матрицы out_q и элемент out_q_0 с номером j на C_p . Аналогичные действия производим с матрицей out_p и вектором out_p_0 , если $out_p[j][j] \neq 0$. Затем, к

каждому элементу строки матриц out_p и out_q (если $out_q[j][j] \neq 0$, $out_p[j][j] \neq 0$), прибавляем его самого, умноженного на $\min[k]$, k - номер столбца элемента. К $out_p_0[j]$, $out_q_0[j]$ прибавляем $out_p[j][i] \cdot C_{\min}$, $out_q[j][i] \cdot C_{\min}$ соответственно.

- Строки с номером j матриц A и B , $a_0[j]$, $b_0[j]$ умножаются на C_p . Затем из каждого элемента j -ой строки вычитается он сам, умноженный на $\min[k]$, k - номер столбца элемента. После из $a_0[j]$ и $b_0[j]$ вычитается $A[j][i] \cdot C_{\min}$, $B[j][i] \cdot C_{\min}$.

Переходим 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:

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

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

Данное преобразование возможно только в том случае, если для данной переменной определена нижняя граница и коэффициенты в матрицах выражений при данной переменной 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 $k = i + 1, \mathbf{K}, m$, $j = 1, \mathbf{K}, m$. Тогда $out_p[k - i][j] = out_p[k][j]$, для матрицы out_q производится аналогичное преобразование. Далее, $out_p[j - i][k] = out_p[j][k]$ и аналогично для out_q .

- Приведение неравенств к новому количеству переменных ($m-1$). Пусть $k = 2, \mathbf{K}, m-1$, $j = 1, \mathbf{K}, m-1$. Имеем $out_p[k-j][j] = out_p[k][j]$, тоже самое для out_q .
- Изменение уравнений. Пусть $k = i+1, \mathbf{K}, m$, $j = 1, \mathbf{K}, n$. Тогда $A[k-1][j] = A[k][j]$, аналогично для матрицы B .
- Приведение уравнений к новому количеству переменных ($m-1$). Пусть $k = 1, \mathbf{K}, m$, $j = 1, \mathbf{K}, n$. Имеем $A[k-j][j] = A[k][j]$, аналогично для B .
- Изменение числа переменных. Уменьшаем число индуктивных переменных на 1, корректируем 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. Тогда 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 проверки выражений на зависимость 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 – сразу стоп.
2. Если система Parent, находящаяся в начале списка, имеет нецелые решения, то строятся новые системы Child_1 и Child_2, содержащие каждая 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) т.к. Parent является первой системой в списке, и последний 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 $i(t) - j(t)$, $(i(t), j(t))$ - 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 в накрутке циклов [5].

12. Интерпретация и использование результатов анализа в целях оптимизации

Результатом анализа является 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 )
            {
                for ( i4 = i1 - i2 + 3 * i3, j2 = i2; i4 < i3 + 5 * i2 - i1; i4++, j2 +=2 )
                {
                    for ( i5 = i4 + i3 + i2 - i1; i5 < i2 + i4 - 3 * i1 + 5 * i3; i5++ )
                    {
                        array[i1][i2+1][i3][i4-3][i5][i3][i2] = array[i1-2][i2+2][i3+1][i4+5][i5][i3-1][i2] + 1;
                        array1[i1][j3-1][i3][i4][i5 + 6][j2][i2] = array1[i1-2][i2+2][i3+1][i4+5][i5][i3-1][i2] + 2;
                    }
                }
            }
        }
    }
}
```

Среднее время работы 10000 запусков алгоритма анализа зависимостей на одинаковых данных приведены в таблицах (время указано в секундах).

Сохранение информации об измерения массива	Целочисленный симплекс-метод	Рациональный симплекс-метод	Метод Фурье
нет	17.0727	8.13923	13.34269
да	16.4688	7.67615	8.03192

Ниже приведены запуски данных алгоритмов на тестах из пакета Spec92, без использования информации о сохранении измерений массивов.

Тест	Целочисленный симплекс-метод	Метод Фурье	Рациональный симплекс-метод
090.hydro2d	0.231413	0.17365	0.130395
089.su2cor	1.07279	0.706621	0.268258

094.fpppp	0.820603	0.53004	0.337288
093.nasa7	3.26257	2.209695	0.715829

Из приведенных таблиц видно, что целочисленный метод работает медленнее. Это объясняется тем, что после решения системы линейных неравенств не всегда получается целочисленный ответ, и его необходимо получать методом Списка Систем Неравенств (описан в разделе 10). Использование целочисленного симплекс-метода предпочтительней, т.к. он дает более точные целочисленные результаты. Метод Фурье оптимизирован для работы с сохранением информации об измерениях массивов, поэтому на данном тесте метод Фурье работает хуже, чем рациональный симплекс-метод. В случае сохранения информации об измерениях массива все три метода работают быстрее, причем метод Фурье работает почти также быстро, как и рациональный симплекс-метод.

14. Заключение

Важным результатом данной работы являются приведенные алгоритмы поиска гнезд циклов, индуктивных переменных, а также используемые алгоритмы решения системы линейных неравенств. Как уже упоминалось, предложенный алгоритм поиска гнезд циклов, снимает многие ограничения, поэтому множество гнезд, к которым можно применить анализ, существенно возрастает. Использование информации об измерениях массивов ускоряет работу алгоритмов решения систем линейных неравенств, особенно метод Фурье. В сравнении с существующими алгоритмами (Power Test [6], Omega Test [7]) анализа зависимостей 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. Tirumalai, 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