МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Государственное образовательное учреждение
высшего профессионального образования
«Ростовский государственный университет»

Механико-математический факультет
Кафедра алгебры и дискретной математики

Магистерская диссертация

Преобразования гнёзд циклов в открытой распараллеливающей системе

Магистрант 2го года Шилов М. В.

Научный руководитель
д.т.н. Штейнберг Б. Я.

Рецензенты
к.ф.-м.н.,
доц. каф. ВМ мехмата РГУ Землянухина Л. Н.

к.ф.-м.н.,
доц. каф. АДМ мехмата РГУ Деундяк В. М.

Ростов-на-Дону
2006 г.

Содержание

Введение

В данной работе рассмотрены оптимизирующие преобразования «перестановка циклов» и «преобразование циклов с помощью неунимодулярных матриц». Приведены условия равносильности перестановки циклов в тесном гнезде. Сформулированы и доказаны условия равносильности в случае нетесного гнезда циклов. Ранее в работе [4] М. Вольфом были рассмотрены несколько иные условия равносильности преобразования «перестановка циклов в нетесном гнезде», но они были представлены без обоснования и описания алгоритма автоматического применения. Также приводятся условия эквивалентности преобразования циклов с помощью неунимодулярных матриц. Написана программная реализация этих преобразований для Открытой Распараллеливающей Системы (ОРС). Оба преобразования автоматизированы. Вся работа по разработке программной реализации ведётся во внутреннем представлении ОРС.

Перестановка циклов - это преобразование, которое изменяет порядок выполнения циклов в гнезде. Тем самым, перестановка циклов является одним из наиболее мощных преобразований и может улучшить характеристики программы по разным причинам:

- Увеличить параллелизм;
- Локальность обращения к памяти;
- Ввести векторизацию цикла;
- Повысить зернистость параллелизма;
- Модифицировать способ доступа к массивам;
- Изменить способ распределение регистров;

Данное преобразование может быть использовано при распараллеливании программ практически на все типы параллельных

компьютеров: векторные, конвейерные, векторно-конвейерные, MIMD, SIMD и т.д. [4][48].

В связи с тем, что данное преобразование не всегда эквивалентно, для проверки эквивалентности требуется выполнение критерия перестановки циклов в тесном двумерном гнезде [3, 47]. В данной работе приводится обобщение критерия перестановки циклов на многомерные гнезда. Для нетесного гнезда циклов сформулированы и доказаны условия эквивалентности перестановки. Данное доказательство основывается на критерии перестановки тесного гнезда циклов в двумерном гнезде.

Преобразование циклов с помощью неунимодулярных матриц используется для конструирования преобразований как: loop interchange (перестановка циклов), skewing (скашивание гнезда циклов), loop reversal (инверсия циклов) [29, 47], hyperplane method (метод гиперплоскостей) [33], tiling or blocking (объединение данных в блоки) [47, 39], rotation (циклический сдвиг) [31].

С помощью неунимодулярных преобразований можно добиться уменьшения требуемых пересылок данных и эффективного использования Кеш памяти [34, 39]. Также данное преобразование используется для систолических алгоритмов [43], для векторизации и распараллеливания циклов [33].

Унимодулярные преобразования являются частным случаем неунимодулярных. Для унимодулярных и для неунимодулярных преобразований необходимо генерировать новые границы гнезда циклов. Для унимодулярных преобразований точные границы нового гнезда циклов находятся легко. Методы генерации новых границ, используемые, для унимодулярных преобразований не работают в случае неунимодулярных преобразований, так как они не могут сгенерировать точные границы нового гнезда циклов. В данной работе представлены методы генерации границ для

неунимодулярных преобразований [28]. Также приводятся условия эквивалентности для неунимодулярных преобразований.

Преобразование перестановка циклов и неунимодулярное преобразование являются частью библиотеки преобразований «Открытой распараллеливающей системы» [11]. Вся работа с преобразованиями ведётся в специальном внутреннем представлении данной системы [14]. Язык реализации всех модулей системы - C++.

Для программной реализации преобразования перестановки циклов были написаны следующие вспомогательные преобразования и условия, которые проверяются автоматически:

- Проверка является ли гнездо циклов тесным или нетесным.
- Проверка линейной зависимости границ циклов в гнезде от счетчиков вышестоящих циклов.
- Проверка эквивалентности преобразования.
- Генерация границ циклов для нового гнезда.

Для программной реализации преобразования циклов с помощью неунимодулярных матриц были написаны следующие вспомогательные преобразования и условия, которые проверяются автоматически:

- Алгоритм Фурье-Моцкина.
- Приведение матрицы преобразования к её Эрмитовой нормальной форме.
- Проверка является ли гнездо циклов тесным или нетесным.
- Проверка линейной зависимости границ циклов в тесном гнезде от счетчиков вышестоящих циклов.
- Проверка эквивалентности преобразования.

Оба преобразования автоматизированы, что естественно упрощает использование данных преобразований.

Автоматическое распараллеливание позволяет [10]:

- ускорить время разработки параллельных программ;
- понизить требования к квалификации программистов, пишущих параллельные программы, поскольку от них можно скрыть многие особенности конкретной параллельной архитектуры;
- повысить надежность разрабатываемых программ, т.к. объем исходного текста для последовательной программы на порядок меньше, чем параллельной, и логическая структура ее проще;
- перенести некоторые разработанные программы с последовательных компьютеров на параллельные системы.

1. Информационная зависимость в программе

Определение. Вхождением переменной будем называть всякое появление переменной в тексте программы вместе с тем местом в программе, в котором эта переменная появилась [10].

Не умаляя общности, все переменные, кроме счетчиков циклов, можно считать индексными. Всякому вхождению при конкретном значении индексного выражения соответствует обращение к некоторой ячейке памяти. Если при этом обращении меняется состояние ячейки (вхождение в левую часть оператора присваивания, которое не входит в индексное выражение другого вхождения), то такое вхождение называется генератором. Остальные вхождения называются использованиями.

Пример 1. В следующем операторе присваивания

A (B (I + 1) = C (2 * I - 2) + A (I + 2)

1 2 3 4

четыре вхождения (их номера проставлены снизу) – и только первое является генератором.

Определение. Два вхождения u, v порождают информационную зависимость, если при некоторых допустимых значениях индексных выражений они обращаются к одной и той же ячейке памяти. При этом v зависит от u, если вхождение u обращается к ячейке памяти раньше, чем вхождение v и

(1) u является генератором, а v – использованием. Зависимость такого типа называется *истинной* или *потоковой зависимостью*, обозначается через δ (u δ v) и иллюстрируется следующей схемой:

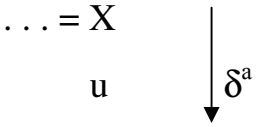
X= . . .

u ↓ δ

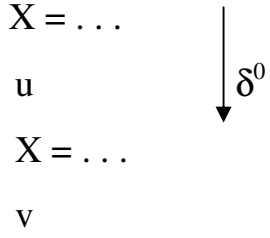
. . . = X

v

(2) u является использованием, а v – генератором. Зависимость такого типа называется *антизависимостью*, обозначается через δ^a (u δ^a v) и иллюстрируется следующей схемой:



(3) u и v являются генераторами. Зависимость такого типа называется *выходной зависимостью*, обозначается через δ^0 (u δ^0 v) и иллюстрируется следующей схемой:



Данное определение есть переформулировка в терминах вхождений определения, приведенного в [1].

Пример 2.

A = B + C

v1 v2 v3

D = A + 2

v4 v5

A = E + D

v6 v7 v8

В данном программном сегменте имеются следующие отношения зависимости: v1 δ v5, v4 δ v8, v5 δ^a v6, v1 δ^0 v6.

Об информационных зависимостях можно прочитать, например, в [8].

Введём обозначение: $\underline{I} = (I_1, I_2, \ldots I_n)$.

Определение. Пусть для векторов $x=(x_1,x_2,...,x_n)$ и $y=(y_1,y_2,..,y_n)$ найдётся число s из $[1, n]_N$, для которого выполняется следующий набор условий: $x_1=y_1, x_2=y_2, .. , x_s<y_s$

В этом случае будем говорить, что вектор y лексикографически старше вектора x. Обозначать этот факт будем так: x<y.

Пример 3.

Пусть x=(1,2,3,8) и y=(1,2,4,6). Вектор y будет лексикографически старше вектора x, поскольку при совпадающих первых двух координатах, третья координата вектора x строго меньше, третьей координаты вектора y. Значения всех последующих координат уже не важны. Для данного примера s=3.

Если вхождение v находится в гнезде *n*-циклов, со счетчиками циклов $I_1, I_2, \ldots I_n$, то будем писать $v[I'_1, I'_2, \ldots I'_n]$ или $v[\underline{I}']$, где $\underline{I}'=(I'_1, I'_2, \ldots I'_n)$. Это делается для того, чтобы сослаться на вхождение v, используемое на итерации цикла, при значениях счетчиков циклов $I_1 = I'_1, I_2 = I'_2, \ldots I_n = I'_n$

1.1. *Векторы направления и расстояния зависимости*

Предположим, что вхождения u и v находятся в гнезде из *n*-циклов со счетчиками циклов $I_1, I_2, \ldots I_n$ соответственно, и предположим, что v зависит от u. Следовательно, существуют такие итерации $\underline{I}$ и $\underline{J}$ ($\underline{I} \le \underline{J}$), что $u[\underline{I}]$ и $v[\underline{J}]$ обращаются к одной ячейке памяти. Между этими итерациями существует отношение $\Omega = (\omega_1, \omega_2,\ldots, \omega_n)$, где $\omega_i \in \{<, =, >\}$. Т.е. если $\underline{I}\ \Omega\ \underline{J}$, то имеют место следующие отношения: $I_i\ \omega_i\ J_i$, для всех i из $[1,n]_N$. Обычно, для зависимых вхождений имеется несколько пар итераций $\underline{I}^k$ и $\underline{J}^k$, на которых

u[$\underline{I}^k$], v[$\underline{J}^k$] обращаются к одной ячейке памяти, где $k \le N = N1*N2*\ldots*Nn$, N1, N2,..Nn – верхние границы циклов n-мерного гнезда. Пусть между парами итераций $\underline{I}^k$ и $\underline{J}^k$ существует отношение Ω^k.

Определение. Объединением отношений Ω^k ($\Omega^k = (\omega_{k,1}, \omega_{k,2},\ldots, \omega_{k,n})$) по k называется отношение $\Psi = \left(\bigcup_k \omega_{k,1}, \bigcup_k \omega_{k,2}, \ldots, \bigcup_k \omega_{k,n}\right)$.

Дополним множество {<, =, >} элементами: {≤} = {<, =}, {≥} = {>, =}, {≠} = {<, >}, {*} = {<, =, >}.

Пример 4. Пусть Ω^1 = (<, >, =), Ω^2 = (=, <, <), Ω^3 = (<, >, >) тогда $\cup\Omega^i$ = {{<, =, <}, {>, <, >}, {=, <, >}}={≤ , ≠ , *} (повторяющиеся элементы в каждом подмножестве были отброшены).

Определение. Вектор направления зависимости (или просто вектор направления) есть объединение по *k* отношений Ω^k по всем парам итераций $\underline{I}^k$ и $\underline{J}^k$, на которых u [$\underline{I}^k$] и v [$\underline{J}^k$] обращаются к одной ячейке памяти.

Данное определение введено по аналогии с определением для расстояния зависимости, приведенным в работе [38].

В таблице 1 приведены сопоставления значений вектора направления зависимости значениям вектора расстояния зависимости.

Таблица 1.

Вектор направления зависимости	Вектор расстояния зависимости
<	Целое положительное число
≤	Целое положительное число

=	0
>	Целое отрицательное число
≥	Целое отрицательное число
≠	Либо отрицательное, либо положительное целое число.
*	Либо отрицательное, либо положительное целое число.

Обозначим вектор направления зависимости через $\Psi = (\psi_1, \psi_2, \ldots \psi_n)$ и будем писать $u\ \delta_{(\psi_1, \ldots \psi_n)}\ v$ или $u\ \delta_{\Psi}\ v$ (где $\psi_i \in \{<, =, >, \geq, \leq, \neq, *\}$).

Замечание. Если имеет место отношение $u\delta_{(\psi_1, \ldots \psi_n)}v$, то между любыми парами итераций $\underline{I}$ и $\underline{J}$, такими что u [$\underline{I}$] и v [$\underline{J}$] обращаются к одной ячейке памяти, существует отношение Ψ: $\underline{I}\ \Psi\ \underline{J}$, т.е. $I_i\ \psi_i\ J_i$, для всех *i* из $[1, n]_N$. Для остальных типов зависимостей вектор направления вводится аналогично.

Пример 5.

```
For i = 0, 10 do
For j = 0, 10 do
  A (2 * i + 3 * j + 21) = …
  u
  . . .    = A (19 * i + 2 * j + 3)
           v
EndFor
```

В данном примере имеется информационная зависимость uδv. Например, на итерации $\underline{I}$ = (0, 0) вхождение u, а на итерации $\underline{J}$ = (0, 9) вхождение v обращаются к элементу A (21). Отношение между элементами $\underline{I}$, $\underline{J}$ равно

(=, <). Далее, на итерации I=(0, 1) вхождение u, а на итерации J=(1, 1) вхождение v обращаются к элементу A (24).

Отношение между элементами I, J равно $(<,=)$. Т. к. вектор направления зависимости равен объединению всех таких отношений, то здесь он, как минимум, равен $(\leq,\leq)$ [38].

1.2. Граф информационной зависимости

Граф информационных зависимостей – ориентированный граф.

Вершины этого графа – вхождения переменных. Между двумя вершинами существует дуга, если соответствующие им вхождения переменных порождают отношение зависимости по данным. Дуга направлена к вершине соответствующей зависимому вхождению. С каждой дугой связывается информация о типе зависимости по данным (потоковая, анти или выходная зависимость).

Пример 6.

```
For i = 1, N
  A (i - 2) = X (2 * i - 1)
  v1          v2
  X (2 * i) = X (2 * i + 3) + A (i)
  v3          v4                  v5
  X (2 * i + 1) = A ( i - 1)
  v6                v7
EndFor
```

Выпишем список всех нетривиальных дуг графа информационных связей этого цикла: v4 - v6, v5 – v1, v6 – v2, v7 – v1.

Свяжем с дугой графа Лампорта, характеризующей зависимость между вхождениями переменных, дополнительную информацию о направлении этой зависимости. Полученный граф называется графом направлений зависимостей.

2. Перестановка циклов

Для большинства архитектур наибольшее время работы программы отнимают циклически повторяющиеся участки. Поэтому в оптимизирующих и распараллеливающих компиляторах наиболее распространены преобразования циклов.

Перестановка циклов - это преобразование, которое изменяет порядок выполнения циклов в гнезде.

2.1 Описание входных данных для преобразования «перестановка циклов»

Рассматриваются циклы следующего вида:

1. Тесное гнездо циклов

$$
\begin{array}{l}
\text{For } I_1 = L_1,\ U_1 \text{ do} \\
\text{For } I_2 = L_2(I_1),\ U_2(I_1) \text{ do} \\
\quad \ldots \\
\text{For } I_n = L_n(I_1,\ldots, I_{n-1}),\ U_n(I_1,\ldots, I_{n-1}) \text{ do} \\
\quad A(P_1(I_1,\ldots, I_n),\ldots, P_n(I_1,\ldots, I_n)) = \ldots \\
\quad \mathbf{u} \\
\qquad \ldots = A(Q_1(I_1,\ldots, I_n),\ldots, Q_n(I_1,\ldots, I_n)) \\
\qquad\quad \mathbf{V} \\
\text{EndFor} \\
\text{EndFor}
\end{array}
\qquad (1)
$$

. . .

EndFor

Здесь:

a. $I_1,..., I_n$ – счётчики циклов.

b. Все L_i и U_i (i = 1..n) – соответственно нижняя и верхняя граница изменения счётчиков циклов. L_1 и U_1 – константы. Все остальные L_i и U_i (i = 2..n) – целочисленные линейные функции.

c. Все P_j и Q_j (j = 1..n) – целочисленные линейные функции.

2. Нетесное гнездо циклов

For 1 $I_1 = L_1, U_1$ do

For $I_2 = L_2 (I_1), U_2 (I_1)$ do

. . .

For $I_n = L_n (I_1,\ldots, I_{n-1}), U_n (I_1,\ldots, I_{n-1})$ do

$A (P_1 (I_1,\ldots, I_n),\ldots, P_n (I_1,\ldots, I_n)) = \ldots$

u

For $J_1 = L_{n+1} (I_1,\ldots, I_n), U_{n+1} (I_1, \ldots, I_n)$ do

For $J_2 = L_{n+2} (I_1,\ldots, I_n, J_1), U_{n+2} (I_1,\ldots, I_n, J_1)$ do

. . .

For $J_m = L_{n+m} (I_1,\ldots, I_n, J_1,\ldots, J_{m-1}), U_{n+m} (I_1,\ldots, I_n, J_1,\ldots, J_{m-1})$ do

$\ldots = A (Q_1 (I_1,\ldots, I_n, J_1,\ldots, J_m),\ldots, Q_n (I_1,\ldots, I_n, J_1,\ldots, J_m))$

V

EndFor

EndFor

. . .

EndFor

Здесь:

a. $I_1,\ldots, I_n, J_1, \ldots, J_m$ – счетчики циклов.

b. Все L_i и U_i (i=1..n+m) – соответственно нижняя и верхняя граница изменения счётчиков циклов. L_1 и U_1 – константы. Все остальные L_i и U_i (i=2..n+m) – целочисленные линейные функции.

c. Все P_j и Q_j (j=1..n) – целочисленные линейные функции.

2.2 Критерий перестановки циклов в тесном двумерном гнезде

Пусть имеется гнездо из двух циклов из класса решаемых задач, следующего вида:

For $I_1 = L_1, U_1$ do

For $I_2 = L_2(I_1), U_2(I_1)$ do

… (2)

$A(P_1(I_1, I_2), P_2(I_1, I_2)) = \ldots$

u

$\ldots = A(Q_1(I_1, I_2), Q_2(I_1, I_2))$

v

. . .

EndFor

EndFor

Теорема 1. Заголовки циклов в гнезде (2) можно переставить тогда и только тогда, когда в графе направлений зависимости нет зависимостей любого вида имеющих вектор направления зависимости $\Psi = (\psi_1, \psi_2)$ такой, что $\{<\} \subset \psi_1$ и $\{>\} \subset \psi_2$ [3, 47].

Замечание 1. Если в данном критерии предполагать, что зависимость между вхождениями u и v препятствует преобразованию (т.е. преобразование пространства итераций гнезда циклов недопустимо), то существует программа, преобразование которой не эквивалентно из-за зависимости между вхождениями u и v [3, 47].

Пример 7. Рассмотрим гнездо циклов:

```
For i=0, 3 do
For j=0, 3 do
  A (i + j) = i + j
  u
EndFor
EndFor
```

Несмотря на то, что вектор направления зависимости (в данном случае имеем выходную зависимость) равен $\{<,>\}$, перестановка заголовков гнезда циклов приводит к эквивалентной программе. Однако мы полагаем, что перестановка циклов недопустима т.к. существует программа, с идентичным вхождением u и с такой же зависимостью, перестановка циклов в которой ведет к неэквивалентной программе.

2.3. Обобщение критерия перестановки на многомерные гнезда циклов

Рассмотрим гнездо циклов (1).

Предположим $u\delta_{\Psi}v$. Вектор направления зависимости Ψ имеет n компонент. Предположим, мы хотим поменять местами заголовки циклов с номерами k и l ($k<l\le n$). Зафиксируем все координаты гнезда циклов, кроме координат I_k и I_l. Пусть $I_1=I'_1$, $I_2=I'_2$, …, $I_{k-1}=I'_{k-1}$, $I_{k+1}=I'_{k+1}$, …, $I_{l-1}=I'_{l-1}$, $I_{l+1}=I'_{l+1}$, …, $I_n=I'_n$. Пусть D – область изменения координат I_k, I_l принадлежит плоскости.

Если при изменении координат I_k и I_l не существует таких итераций $\underline{I} = (I'_1, \ldots, I'_{k-1}, I_k, I'_{k+1}, \ldots, I'_n)$ и $\underline{J} = (I'_1, \ldots, I'_{l-1}, I_l, I'_{l+1}, \ldots, I'_n)$, что u[$\underline{I}$] и v[$\underline{J}$] обращаются к одной ячейке памяти, то при этом фиксированном наборе координат область D можно обходить в любом направлении (независимо от компонент вектора направления зависимости ψ_k, ψ_l). В этом случае вхождения u и v обращаются к одним и тем же ячейкам памяти при разных фиксированных наборах координат, т.е. эти фиксированные наборы координат определяют порядок обращения вхождений к одним и тем же ячейкам памяти.

Если при изменении координат I_k и I_l существуют такие итерации $\underline{I} = (I'_1, \ldots, I'_{k-1}, I_k, I'_{k+1}, \ldots, I'_n)$ и $\underline{J} = (I'_1, \ldots, I'_{l-1}, I_l, I'_{l+1}, \ldots, I'_n)$, что u[$\underline{I}$] и v[$\underline{J}$] обращаются к одной ячейке памяти. Тогда изменение обхода области D может изменить порядок обращения вхождений u и v к элементу памяти u[I] (он же v[J]). В этом случае необходимо применять критерий перестановки циклов к компонентам вектора направления зависимости ψ_k, ψ_l построенный при фиксированном наборе координат $I'_1, I'_2, \ldots, I'_{k-1}, I'_{k+1}, \ldots, I'_{l-1}, I'_{l+1}, \ldots, I'_n$.

Фиксируя некоторый набор координат, мы, фактически, переходим к случаю анализа возможности перестановки двух циклов. А именно, анализ, описанный в начале этого параграфа. Т.к. теперь все значения координат I_k и I_l, при которых вхождения u и v обращаются к одним и тем же ячейкам памяти, принадлежат области D. Изменение обхода области D может изменить порядок обращения вхождений u и v к одним и тем же ячейкам памяти.

Для указанного выше фиксированного набора координат и координат I_k и I_l построим, по определению, вектор направления зависимости $\Psi' = (*, *, \ldots, *, \psi_k, *, \ldots, *, \psi_l, *, \ldots, *)$. Данный вектор направления Ψ' будет описывать только те зависимости, которые возникают при изменении координат I_k и I_l в области D. Ясно, что $\Psi' \subset \Psi$ [16].

Теорема 2. Зависимость между u и v препятствует перестановке циклов k и l, если существует фиксированный набор координат $I'_1, I'_2, \ldots, I'_{k-1}, I'_{k+1}, \ldots, I'_{l-1}, I'_{l+1}, \ldots, I'_n$, и вектор направления зависимости $\Psi' = (*, *, \ldots, *, \psi_k, *, \ldots, *, \psi_l, *, \ldots, *)$, построенный для этого фиксированного набора, такой, что $\psi_k = \{<, \leq\}$ и $\psi_l = \{>, \geq\}$ [16].

2.4 Алгоритм преобразования «перестановка циклов» в тесном гнезде

- Распознавание применимости исходного гнезда циклов

Программно реализованы следующие проверки:

1. Являются ли границы циклов в гнезде линейно - зависимыми от счетчиков внешних циклов. Для проверки этого реализована функция hasLinearityBounds (StmtFor* Loop), которая получает гнездо циклов во внутреннем представлении и проверяет,

является ли его границы линейно – зависимыми от счетчиков внешних циклов. Делается разбор выражений, которыми являются границы циклов, и проверяется линейность выражений. Если границы не являются линейно – зависимыми от счетчиков внешних циклов, то на экран выводится сообщение, что цикл имеет границы отличные от линейных.

2. Является ли исходное гнездо циклов тесным или нетесным. Для проверки этого реализована функция isPerfectlyNested (StmtFor* Loop), которая получает исходное гнездо циклов во внутреннем представлении и проверяет, существуют ли между заголовками циклов в гнезде операторы отличные от заголовков циклов “For”. Если найдены такие операторы, то на экран выводится сообщение, что гнездо циклов является нетесным.

3. Является ли шаг каждого цикла в исходном гнезде равным единице. Для проверки этого реализована функция hasStepEqualUnit (StmtFor* Loop), которая получает исходное гнездо циклов и проверяет шаг каждого цикла в гнезде на равенство единицы. Если найден шаг отличный от единицы, то на экран выводится сообщение, что в гнезде циклов имеется цикл с шагом отличным от единицы.

4. Эквивалентность перестановки циклов. Для проверки этого реализована функция isEquivalenceConversion (StmtFor* Loop), которая получает исходное гнездо циклов и делает необходимые манипуляции. После этого вызывается специальная функция LoopsAreInterchangableEx (), которая проверяет условие критерия перестановки циклов в тесном гнезде, и если оно выполнено, то переставляет $k^{ий}$ с $(k+1)^{ым}$ заголовком циклов. Для реализации алгоритма, проверки условия теоремы перестановки циклов в

тесном гнезде, был использован решётчатый граф программы. Данная реализация написана аспирантом кафедры «Алгебры и Дискретной Математики» Шульженко А. М. [16].

- Генерируются новые границы преобразованного гнезда циклов.

Перед генерацией границ, из гнезда циклов представленного во внутреннем представлении извлекаются старые границы в специальные структуры. Для левой и правой границы каждого заголовка циклов такой структурой является LinearExpression. В ней хранятся линейные выражения. Для заголовка цикла такой структурой является LoopBounds. В ней хранится пара: левая и правая границы. Для гнезда циклов такой структурой в программе является LoopBoundsNode. В ней хранятся заголовки циклов в той же последовательности что и в исходном гнезде циклов.

Для конвертирования границ из внутреннего представления в структуры, описанные выше и обратно, были написаны специальные функции:
Функция LinearExpressionToExprNode () – переводит линейное выражение из структуры LinearExpression во внутреннее представление.

Функция LoopBoundsToExprNode () – переводит левые и правые границы из структуры LoopBounds во внутреннее представление.

Функция LoopBoundsNodeToExprNode () – переводит гнездо циклов из структуры LoopBoundsNode во внутреннее представление.

Функция ExprNodeToLinearExpression () – переводит линейное выражение представленное во внутреннем представлении в структуру LinearExpression.

Функция ExprNodeToLoopBounds () – переводит левые и правые границы заголовка цикла гнезда, представленные во внутреннем представлении, в структуру LoopBounds.

Функция ExprNodeToLoopBoundsNode () – переводит гнездо циклов, представленное во внутреннем представлении, в структуру LoopBoundsNode.

После того как, сформирована структура LoopBoundsNode по исходному гнезду циклов, она используется для генерации новых границ. Для генерации новых границ использовался алгоритм [23], вместо алгоритма [18], по следующим причинам:

1. как показано в работе [21], алгоритм [23] требует меньше памяти, чем алгоритм [18];
2. как показано в работе [21], при больших размерностях гнезд циклов алгоритм [23] выполняется быстрее, чем алгоритм [18];
3. алгоритмом [18] можно сгенерировать только сразу все границы всех циклов в гнезде; алгоритм [23] более удобен, им можно генерировать границы циклов отдельно для каждой глубины вложенности;

После того как новые границы сгенерированы, они переводятся во внутреннее представление, и делается замена старых границ на новые.

2.5 Вспомогательные преобразования для обоснования перестановки циклов в нетесном гнезде.

2.5.1. Раскрутка цикла.

Раскрутка цикла [10] заменяет цикл

```
For I=1, N do
   ТЕЛОЦИКЛА (I)
EndFor
```

на следующий фрагмент программы

```
ТЕЛОЦИКЛА(1)
ТЕЛОЦИКЛА(2)
   . . .
ТЕЛОЦИКЛА(N)
I = N
```

Границы цикла должны быть константами (в частности, значение N в данном цикле должно быть известно на этапе трансляции).

Если ТЕЛОЦИКЛА(I) содержит помеченные операторы, на которые указывают goto, то в копиях тела цикла метки должны вводиться новые. В упрощенном варианте раскрутку можно применять лишь для циклов, в теле которых не содержаться помеченные операторы, на которые указывают goto.

Счетчику цикла следует присвоить значение последней итерации.

Результирующий фрагмент программы должен быть в операторных скобках. Это важно для того случая, если бы исходный цикл был телом некоторого другого объемлющего цикла или в зоне действия условного оператора.

Поскольку при раскрутке цикла не меняется порядок выполнения операций, это преобразование всегда эквивалентно.

2.5.2. Разбиение цикла

Разбиение цикла [10] заменяет цикл

```
For I = 1, N do
S1
.....
Sk
S(k+1)
.....
Sm
EndFor
```

на фрагмент программы, состоящий из последовательности двух циклов

```
For I = 1, N do
S1
 .  .  .
Sk
EndFor

For I = 1, N do
S(k+1)
 .  .  .
Sm
EndFor
```

Теорема 3. Пусть выполнены следующие условия:

1) каждый из фрагментов программы S1,...,Sk и S(k+1),...,Sm имеет один вход и один выход. В частности:

1.1. ни один из операторов S(k+1),...,Sm не находится в зоне действия какого либо условного оператора из множества S1,...,Sk. (т.е. выполнение или невыполнение этого оператора не зависит от значения логического выражения условного оператора)

1.2. ни один из операторов S(k+1),...,Sm не находится в теле цикла, заголовок которого во множестве S1,...,Sk

1.3. всякий оператор goto из множества S1,...,Sk (или S(k+1), ..., Sm) указывает на метку оператора из этого же множества.

2) не существует такой дуги графа информационных связей (v1,v2), что v1 принадлежит Si, k+1=<i=<m, v2 принадлежит Sj, 1=<j=<k.

Тогда разбиение циклов является эквивалентным преобразованием.

Замечание. Результирующий фрагмент программы должен быть в операторных скобках. Это важно для того случая, если бы исходный цикл был телом некоторого другого объемлющего цикла или под действием условного оператора.

2.5.3 Слияние циклов

Данное преобразование заменяет фрагмент программы, состоящий из двух подряд написанных циклов с одинаковыми заголовками, одним циклом (loop fusion) [10].

Это преобразование является обратным к разбиению циклов.

Условие выполнения: если после выполнения преобразования получается цикл, который можно, сохраняя равносильность, разрезать на две части, получив исходный фрагмент, то данное преобразование равносильно. Кроме того, есть ограничение, связанное с синтаксической корректностью. В исходной программе не должно быть переходов на заголовок второго цикла, поскольку в этом случае в результирующей программе этот переход исчезнет

или возникнет недопустимый синтаксисом переход извне во внутрь цикла. И еще, при взятии в операторные скобки обоих исходных циклов должна получаться программа эквивалентная исходной. Это условие может не выполняться, если первый из исходных циклов является телом некоторого третьего объемлющего цикла, а второй исходный цикл – не является.

2.5.4. Перестановка операторов присваивания

Данное преобразование состоит в том, чтобы фрагмент программы

```
S1
.....
S(k-1)
Sk
S(k+1)
.....
Sm
```

заменить на фрагмент программы

```
S(k+1)
.....
Sm
S1
.....
S(k-1)
Sk
```

Здесь S1,...,Sm - операторы программы.

Теорема 4 . Пусть для фрагментов программы S1,...,Sk и S(k+1),...,Sm операторы S1 и S(k+1) являются единственными точками входа, а операторы Sk и Sm - единственными точками выхода

соответственно. Кроме того, будем предполагать, что для фрагмента S1,...,Sk,S(k+1),...,Sm оператор S1 является единственным входом, а Sm - единственным выходом (это условие не вытекает из предыдущего). Предположим, что в графе информационных связей НЕ существует такой циклически независимой дуги (v1,v2), что v1 принадлежит одному из переставляемых фрагментов программы, v2 - другому. (Циклическая независимость означает, что для некоторого момента выполнения всей последовательности операторов S1,...,Sk, S(k+1),...,Sm оба вхождения v1 и v2 обращаются к одной и той же ячейке памяти). Тогда рассматриваемая перестановка фрагментов программы синтаксически корректна и эквивалентна.

2.6. Условия перестановки циклов в нетесном гнезде

Перестановка циклов в нетесном гнезде представляет собой замену фрагмента (3)

```
For I = 1, N do
LOOPBODY1 (I)          (3)
For J = 1, N do
LOOPBODY2 (I, J)
EndFor
EndFor
```

на фрагмент (4)

```
For J = 1, N do
LOOPBODY1 (J)          (4)
For I = 1, N do
LOOPBODY2 (I, J)
```

EndFor

EndFor

И рассмотрим вспомогательное гнездо циклов (5), которое может быть получено из (3) удалением фрагмента программы LOOPBODY1:

For I = 1, N do

For J = 1, N do

LOOPBODY2 (I, J) (5)

EndFor

EndFor

Здесь LOOPBODY1, LOOPBODY2 - фрагменты программы, состоящие из операторов присваивания.

Определение. Два фрагмента программы эквивалентны тогда и только тогда, когда при одних и тех же исходных данных на их выходе получаются одинаковые значения результирующих данных [10].

Теорема 6. (Перестановка циклов в нетесном гнезде)

Пусть для фрагмента (3) выполняются следующие условия:

1. каждый из фрагментов программы LOOPBODY1 и LOOPBODY2 имеет один вход и один выход.
2. не существует такой дуги (v1, v2), что v1 принадлежит LOOPBODY2, v2 принадлежит LOOPBODY1 или v1 принадлежит LOOPBODY1, v2 принадлежит LOOPBODY2.
3. Левые и правые границы внешнего цикла и внутреннего цикла совпадают соответственно. Шаг внешнего цикла совпадает с шагом внутреннего цикла и равен единице.

4. Гнездо циклов (5) удовлетворяет условиям перестановки циклов в тесном гнезде.

Тогда перестановка циклов нетесного гнезда является эквивалентным преобразованием.

Доказательство: Из условий теоремы получаем, что для гнезда циклов (3) выполнены все условия теоремы «разбиения циклов». Таким образом, применяя преобразование "разбиение циклов" для гнезда (3) получаем, что оно преобразуется к фрагменту программы, состоящему из двух гнёзд циклов: одномерному и двумерному

а)
```
For I = 1, N do
LOOPBODY1 (I)
EndFor
```

б)
```
For I = 1, N do
For J = 1, N do
LOOPBODY2 (I, J)
EndFor
EndFor
```

Из условия 4 следует, что в гнезде б) циклы можно переставить. Таким образом, получим:

в)
```
For J = 1, N do
For I = 1, N do
LOOPBODY2 (I, J)
EndFor
EndFor
```

И так как изменение имени счетчика цикла во всем цикле, включая тело цикла, является эквивалентным преобразованием. Тогда можно заменить счетчик I в цикле а) на J, после чего цикл а) примет вид:

г) For J = 1, N do
LOOPBODY1 (J)
EndFor

Циклы в) и г) удовлетворяют условию теоремы слияния циклов. Таким образом, применив преобразование «слияние циклов» для в) и г) мы получим цикл (4) эквивалентный (3).
Теорема доказана.

Теорема 7. Пусть выполнены условия критерия перестановки циклов в тесном гнезде (5). Если два фрагмента программы (3) и (4) не эквивалентны, то во фрагменте (3) существует дуга (v1, v2), такая что v1 принадлежит LOOPBODY2, а v2 принадлежит LOOPBODY1 или v1 принадлежит LOOPBODY1, а v2 принадлежит LOOPBODY2.
Доказательство: Раскрутим циклы фрагментов (3) и (4):

```
For I = 1, N do
LOOPBODY1 (I)          (3)
For J = 1, N do
LOOPBODY2 (I, J)
EndFor
EndFor
```

```
For J = 1, N do
LOOPBODY1 (I)          (4)
For I = 1, N do
LOOPBODY2 (I, J)
EndFor
EndFor
```

Раскрутка фрагмента (3)

```
LOOPBODY1 (1)
LOOPBODY2 (1, 1)
LOOPBODY2 (1, 2)
    . . .
LOOPBODY2 (1, N)
```

Раскрутка фрагмента (4)

```
LOOPBODY1 (1)
LOOPBODY2 (1, 1)
LOOPBODY2 (2, 1)
    . . .
LOOPBODY2 (N, 1)
```

```
LOOPBODY1 (2)
LOOPBODY2 (2, 1)
LOOPBODY2 (2, 2)          (6)
    . . .
LOOPBODY2 (2, N)
```

```
LOOPBODY1 (2)
LOOPBODY2 (1, 2)
LOOPBODY2 (2, 2)          (7)
    . . .
LOOPBODY2 (N, 2)
```

```
 . . .
LOOPBODY1 (N)
LOOPBODY2 (N, 1)
LOOPBODY2 (N, 2)
    . . .
LOOPBODY2 (N, N)
```

```
      . . .
LOOPBODY1 (N)
LOOPBODY2 (1, N)
LOOPBODY2 (2, N)
    . . .
LOOPBODY2 (N, N)
```

Фрагмент (6) получен из фрагмента (7) после перестановки операторов присваивания. Пусть фрагменты (6) и (7) не эквивалентны. Следовательно, найдется некоторая переменная, обозначим ее A, принимающая разные значения после выполнения этих фрагментов. Это возможно лишь в двух случаях:

1) последнее значение переменной A в этих фрагментах вычисляют разные операторы;

2) последнее значение переменной A вычисляет один и тот же оператор, но этот оператор использует разные значения некоторой переменной, которую будем обозначать B.

По теореме о перестановке операторов присваивания, имеем дугу либо *in-out*, либо *out-in*, либо *out-out*, которая препятствует перестановке операторов присваивания. Учитывая условия теоремы о перестановке циклов в тесном гнезде (5), дуги, принадлежащие только LOOPBODY2, не препятствуют перестановке операторов присваивания во фрагменте (3). Следовательно, данная дуга, является дугой между LOOPBODY1 и LOOPBODY2.

Теорема доказана.

В работе М. Вольфа [4], рассмотрены условия перестановки циклов в нетесном гнезде, но эти условия приводятся без обоснования.

В статье рассмотрены циклы вида:

```
DO 100 I = L1, U1
  S1(I)  =  …
DO 100 J = L2, U2
  S2(I, J) = …
```

Условия эквивалентности, рассмотренные М. Вольфом, имеют вид:

$S1\ (I) \rightarrow S1\ (I')$ $\quad I < I'$

$S2\ (I, J) \rightarrow S2\ (I', J')$ $\quad I < I', J < J'$

$S1\ (I) \rightarrow S2\ (I', J')$ $\quad I < I', I < J'$

$S2\ (I, J) \rightarrow S1\ (I')$ $\quad I < I'\ \ J < I'$

где $\rightarrow$ означает потоковую зависимость.

Автором статьи утверждается, что при выполнении этих условий перестановка циклов в нетесном гнезде приведёт к эквивалентному гнезду циклов. Также в этой статье приводится пример двумерного гнезда циклов следующего вида:

```
        DO 100 I = K + 1; N
S1              A (I, K) = A (I – 1, K + 1) * A (K + 1, K + 1)
        DO 100 J = K + 1, N                                          (B)
S2      100     A (I, J) = A (I, J) + A (I, K) * A (K, J)
```

Гнездо циклов (B) удовлетворяет условиям перестановки циклов, рассмотренные М. Вольфом. Гнездо циклов (C) автором получено после перестановки циклов в гнезде (B):

```
        DO 100 J = K + 1; N
S1              A (J, K) = A (J – 1, K + 1) * A (K + 1, K + 1)
```

```
      DO 100 I = K + 1, N                                    (C)
S2    100   A (I, J) = A (I, J) + A (I, K) * A (K, J)
```

В результате делается заключение, на основании приведённых выше условий, что гнезда циклов (B) и (C) эквивалентны.

Для опровержения данного заключения рассмотрим начальные данные для массива A и внешних переменных K и N:

Пусть K = 0, N = 3 и значения массива A заданны в виде матрицы:

$$A = \begin{pmatrix} 2 & 1 & 2 \\ 2 & 3 & 4 \\ 5 & 6 & 7 \end{pmatrix}$$

Сравнив результаты, полученные после работы циклов (B) и (C), получим:

При работе цикла (B)

$$A^1 = \begin{pmatrix} 2 & 1 & 2 \\ 4 & 8 & 13 \\ 64 & 66 & 131 \end{pmatrix}$$

При работе цикла (C)

$$A^2 = \begin{pmatrix} 2 & 1 & 2 \\ 4 & 8 & 13 \\ 64 & 9 & 131 \end{pmatrix}$$

Матрицы A^1 и A^2 различаются в элементе a_{32}. Таким образом, фрагменты программ (B) и (C) не эквивалентны, так как при одинаковых входных данных, получаем различные выходные данные.

Применяя теорему 6, доказанную в данной работе, становится понятно, что переставлять циклы в гнезде (B) нельзя из-за потоковой зависимости между операторами S1 и S2.

2.7. Примеры использования перестановки циклов

Применение перестановки циклов для многоконвейерной архитектуры

.

Предположим, что используется многоконвейерная архитектура [10]. Тогда может случиться, что в тесном гнезде циклов после перестановки циклов эффективность кода повысится.

Пример 8.

```
For i = 2, N do
For j = 2, M do
   X (i, j) = X (i - 1, j) + A (i, j)
EndFor
EndFor
```

При двумерной конвейеризации исходного цикла конвейеры следует запускать со сдвигом см. [Ш.Р.]. Но если предварительно заголовки циклов поменять местами, то все конвейеры можно будет запустить одновременно.

```
For j = 2, N do
For i = 2, N do
X (i, j) = X (i-1, j) + A (i, j)
EndFor
EndFor
```

Более того, в таком виде конвейеры в процессе работы должны считывать только переменные A (i, j), тогда как в исходном случае пришлось бы считывать и X (i - 1, j). Уменьшение обращений к памяти вдвое может

позволить организацию большего числа одновременно работающих конвейеров.

Пример 9. В данном примере перестановка циклов не влияет на сдвиги в запуске конвейеров, но, учитывая тот факт что, инициализация конвейеров обходится очень дорого, получаем что, переставляя циклы, уменьшается количество таких запусков (но каждый конвейер будет дольше работать) [Ш.Р.].

```
For i = 2, 1000 do
For j = 2, 100 do
X (i, j) = X (i - 1, j) + X (i, j - 1)
EndFor
EndFor

For j = 2, 100 do
For i = 2, 1000 do
X (i, j) = X (i - 1, j) + X (i, j - 1)
EndFor
EndFor
```

Применение перестановки циклов для оптимизации иерархии памяти

Определение. Дистанция по адресам памяти при последовательных к ней обращениях называется страйдом[1].

Данная характеристика показывает эффективность выполнения цикла. Например, если цикл обращается в память через каждые 4 элемента, это цикл со страйдом-4, т.е. разница между соседними итерациями, на которых происходит обращение в память, равна 4. Страйд-1 означает, что цикл

[1] stride – шаг по индексу (при считывании элементов многомерного массива)

обращается в память за каждым следующим по адресу элементом. Страйд-1 наиболее желателен, поскольку он делает максимальной локальность, а, следовательно, и эффективность КЭШа, буфера трансляции адресов TLB[2] и систем страничной памяти. Это также исключает конфликты в банках памяти векторных машин [49].

Буфер TLB предназначен сохранять преобразованные адреса, которые получились в результате трансляции виртуального адреса в физический. Вероятность нахождения адреса в буфере близка к 99%. Такая буферизация очень выгодна, так как если центральный процессор не найдет адрес в TLB, то ему придется вычислять этот адрес. Для разрешения одного адреса процессор должен выполнить три такта. Если умножить три такта на число адресов, к которым процессор должен обратиться, то становится ясно, почему вычисление адресов существенно замедляет процессор. Если же адрес будет существовать в TLB, то он будет разрешен за один такт. Таким образом, использование буфера увеличивает производительность на 200%. [50].

Пример 10.

(a) Исходный цикл

```
For i = 1, n do
For j = 1, n do
    Total [i] = Total [i] + a[i, j]
EndFor
EndFor
```

(b) Гнездо с переставленными циклами

```
For j = 1, n do
```

[2] TLB (Translation Lookaside Buffer) - буфер быстрого преобразования адреса.

```
For i = 1, n do
      Total[i] = Total[i] + a[i,j]
EndFor
EndFor
```

В гнезде циклов (a) внутренний цикл обращается к массиву со страйдом *n* (в Фортране обращение к массиву производится по столбцам). Замена циклов позволяет преобразовать внутренний цикл к страйду-1, как показано для гнезда циклов (b):

Для больших массивов, в которых только часть столбца помещается в КЭШе, эта возможность сильно уменьшает количество промахов КЭШа.
Для векторных архитектур преобразованный цикл дает возможность векторизации благодаря исключению зависимостей на Total[*i*] во внешнем цикле.

Необходимо следить, чтобы возможные выгоды при использовании перестановки циклов не отменили друг друга. Например, обмен, который улучшает использование регистров, может изменить страйд-1 доступа к памяти на страйд-n, который сильно ухудшает общие характеристики вычислений из-за промахов КЭШа.

Рис.1 демонстрирует значительное влияние различных страйдов на скорость вычислений. Данные взяты из работы [49].

Пример 11.

```
For i = 1, 1024∗stride, stride do
  a[i] = a[i] + c
EndFor
```

Цикл с изменяемым страйдом

Страйд	Промахи КЭШ	Промахи TLB	Относительная скорость %
1	64	2	100
2	128	4	83
4	256	8	63
8	512	16	40
12	768	24	28
16	1024	32	23
64	1024	128	19
256	1024	512	12
512	1024	1024	8

Рис.1. Пример влияния страйда на характеристики цикла, приведенного выше

Использование перестановки циклов для векторизации программ

Некоторые циклы на векторных машинах могут выполняться с помощью одной (векторной) команды, т.е. тело цикла выполняется одновременно для всех итераций.

Программный цикл является векторизуемым, если в его теле нет циклически зависимых зависимостей, т.к. в противном случае следующие итерации нельзя выполнять до того, как не будут выполнены предыдущие. Кроме того, операторы тела цикла не должны содержать вызовов функций.

Пример 12

```
For  I =0; 1000 do
  A[i] = B[i] + 2 * i;
EndFor
```

Данный цикл векторизуемый и может быть заменен одной векторной командой.

Пример 13

```
For  I =0; 1000 do
 A[i]=A[i-1]*2;
EndFor
```

Данный цикл не векторизуемый, так как оператор присваивания в его теле рекуррентен (нельзя посчитать, например, вторую итерацию цикла до того, как была выполнена первая).

Пример 14. Перестановка циклов может использоваться для облегчения векторизации программ. Рассмотрим два фрагмента программ

```
For J = 1, N do
  For I = 2, N do
    A (I, J) = A (I – 1, J) + B (I)                    (8)
  EndFor
EndFor
```

И эквивалентный (1) фрагмент

```
For I = 2, N do
  For J = 1, N do
    A (I, J) = A (I – 1, J) + B (I)                    (9)
  EndFor
EndFor
```

В двумерном гнезде (8) внутренний цикл вычисляет линейную рекуррентность; в результате перестановки, приводящей к фрагменту (9), становится возможным векторизация цикла по J [4].

3. Преобразование циклов с помощью неунимодулярных матриц

Преобразование циклов с помощью неунимодулярных матриц используются для моделирования оптимизирующих и распараллеливающих преобразований таких как: loop interchange (перестановка циклов), skewing (скашивание гнезда циклов), loop reversal (инверсия циклов) [29, 47], hyperplane method (метод гиперплоскостей) [33], tiling or blocking (объединение данных в блоки) [47, 39], rotation (циклический сдвиг) [31].

С помощью неунимодулярных преобразований можно добиться уменьшения требуемых пересылок данных и эффективного использования Кеш памяти [34, 39]. Также данное преобразование используется в систолических алгоритмах [43].

Унимодулярные преобразования являются частным случаем неунимодулярных. Для унимодулярных и для неунимодулярных преобразований необходимо генерировать новые границы гнезда циклов. Для унимодулярных преобразований точные границы нового гнезда циклов находятся легко. Методы генерации новых границ, используемые, для унимодулярных преобразований не работают в случае неунимодулярных преобразований, так как они не могут сгенерировать точные границы нового гнезда циклов.

3.1. Описание входных данных для неунимодулярного преобразования циклов

Рассматриваются только тесные гнезда циклов вида:

For $i_1 = L_1, U_1$ do

For $i_2 = L_2(i_1), U_2(i_1)$ do (10)

. . .

For $i_n = L_n(i_1,\ldots, i_{n-1})$, $U_n(i_1,\ldots, i_{n-1})$ do

A [$f_1(i_1,\ldots, i_n), f_2(i_1,\ldots, i_n), \ldots, f_n(i_1,\ldots, i_n)$]

EndFor

. . .

EndFor

EndFor

Здесь:

a. Шаг каждого цикла равен 1
b. L_i и U_i (i=1...n) – соответственно нижняя и верхняя границы изменения счётчиков циклов. Все L_i и U_i (i = 1..n) – целочисленные линейные функции от счетчиков циклов такие, что:

   $L_i = \sum_{k=1}^{i-1} a_i^k * i_k + \alpha_i$ $U_i = \sum_{k=1}^{i-1} d_i^k * i_k + \beta_i$ где d_i^k, a_i^k - $1 \le k \le i-1$ $(i = 1... n)$ – константы, α_i, β_i - могут быть константами или внешними переменными.
c. Функция f – целочисленная линейная функция от счетчиков циклов.

3.2. Пространство итераций гнезда циклов

Определение. Пространством итераций гнезда циклов (*) будем называть множество целых точек принадлежащих многограннику:

$IS = \{I = (I_1, I_2, \ldots, I_n)^t \mid L_1 \le I_1 \le U_1; L_2 \le I_2 \le U_2 ; \ldots ; L_n \le I_n \le U_n \}$

Пример 15. Рассмотрим двумерное гнездо циклов:

(a)

For $i_1 = 0, 4$ do

For $i_2 = 0, 6$ do

A $[i_1, i_2]$ = A $([i_1 - 1, i_2])$

EndFor

EndFor

Пространство итераций для данного гнезда циклов будет иметь следующий вид:

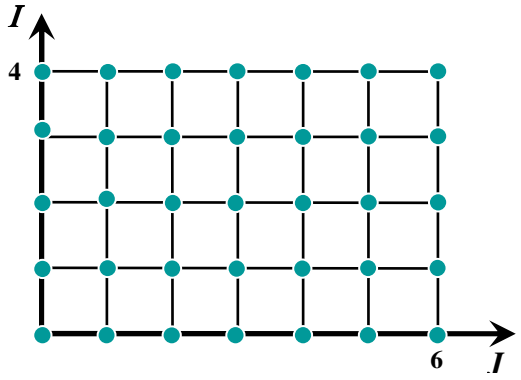


Рис. 1. Пространство итераций гнезда циклов (а)

3.3. Описание пространства итераций в матричной форме

Пространство итераций гнезда циклов (*) может быть описано в матричной форме:

Учитывая что $\sum_{k=1}^{i-1} a_i^k * i_k + \alpha_i \le i_i \qquad i_i \le \sum_{k=1}^{i-1} d_i^k * i_k + \beta_i$. Тем самым, объединяя эти два неравенства, получим: $\sum_{k=1}^{i-1} a_i^k * i_k + \alpha_i \le \sum_{k=1}^{i-1} d_i^k * i_k + \beta_i$ или

$\sum_{k=1}^{i-1} (a_i^k - d_i^k) * i_k \le \beta_i - \alpha_i$. Таким образом, можно записать пространство итераций гнезда циклов в виде $A*I \le \beta$

Пример 16. Для двумерного гнезда циклов определённого в примере 15 матричная форма будет иметь вид:

$$0 \le i_1 \;\; i_1 \le 4$$

$$0 \le i_2 \;\; i_2 \le 6$$

Матричная форма:

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix} * \begin{pmatrix} i_1 \\ i_2 \end{pmatrix} \le \begin{pmatrix} 4 \\ 6 \\ 0 \\ 0 \end{pmatrix}$$

3.4. Матрица преобразования

Для неунимодулярного преобразования требуется матрица следующего вида:

$T = (\mathrm{a}_{i,j})_{i=1}^{n}{}_{,j=1}^{n} \in M_{n\times n}(Z)$ - невырожденная матрица, где $M_{n\times n}(Z)$ - множество матриц порядка $n \times n$ с целыми элементами.

Матрица подбирается в зависимости от преобразования, которое мы хотим получить.

Определение Невырожденная матрица T называется унимодулярной, если её определитель равен ± 1.

Теорема 8. Если матрица T - унимодулярная, то T^{-1} – также унимодулярная. Доказательство непосредственно следует из свойств определителя.

Если определитель невырожденной матрицы T отличен от ± 1, то такую матрицу называют неунимодулярной. В этом случае элементы матрицы T^{-1} принадлежат множеству рациональных чисел.

Таким образом, получается что, если матрица преобразования T – унимодулярная, то T и T^{-1} переводят целочисленные векторы в целочисленные векторы. Если же матрица T – неунимодулярная, то T переводит целочисленные векторы в целочисленные, а T^{-1} переводит целочисленные векторы в рациональные.

3.5. Преобразование пространства итераций матрицей преобразования

Пусть есть матрица преобразования T (унимодулярная или неунимодулярная) и пространство итераций гнезда циклов (10) в матричной форме, т.е. $A*I \le \beta$. Тогда можно "подействовать" матрицей преобразования T на пространство итераций гнезда циклов, другими словами преобразовать пространство итераций гнезда циклов с помощью матрицы преобразования. Для этого из неравенств:

$$A*I \le \beta$$

$$T* I = J$$

Учитывая, что матрица T – обратима, можно получить следующие неравенства:

$$T^{1}* J = I$$

$$A* T^{1}* J \le \beta$$

заменим $A* T^{1}$ на A', получим:

$A' * J \le \beta$ – *преобразованное пространство итераций в матричной форме.*

Пример 17. Рассмотрим пространство итераций, показанное в примере 15, и подействуем на него двумя матрицами T_1 и T_2:

$T_1 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ - унимодулярная матрица $\det T_1 = 1$

$T_2 = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$ - неунимодулярная матрица $\det T_2 \neq 1$

На рис. (2a) показано пространства итераций, которое получено в результате действия матрицы T_1 на пространство итераций из примера 15, на рис. (2b) соответственно для матрицы T_2. В этих схемах только белые и синие точки имеют целочисленные прообразы. Остальные точки являются так называемыми «дырами», т.е. точки которые имеют рациональные прообразы.

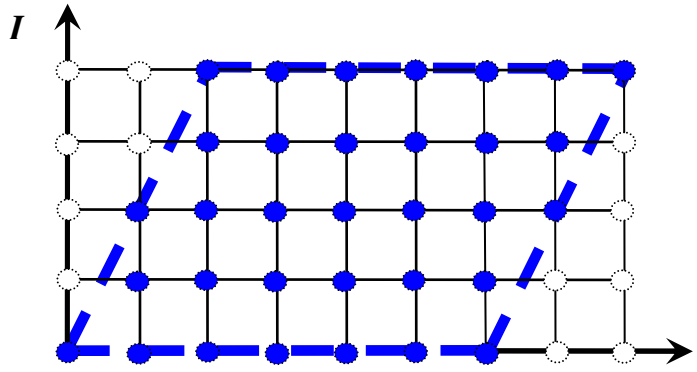


Рис. (2a) Пространство итераций, полученное в результате действия матрицы T_1

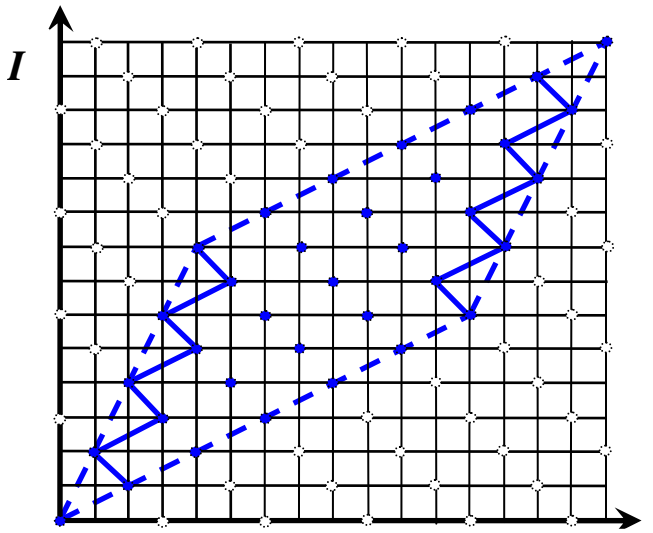


Рис. (2b) Пространство итераций, полученное в результате действия матрицы T_2

При этом если матрица преобразования унимодулярная, то новые границы, которые находятся с помощью алгоритма Фурье-Моцкина по

преобразованному пространству итераций, будут точными и могут быть непосредственно использованы для построения нового гнезда циклов. Если же матрица преобразования неунимодулярная, то преобразованное пространство итераций будет содержать дыры. Это следует из того, что границы нового гнезда циклов, полученные с помощью алгоритма Фурье-Моцкина по преобразованному пространству итераций, получаются неточными. Эти точки (дыры) должны быть пропущены при построении нового гнезда циклов.

3.6. Условия эквивалентности неунимодулярного преобразования циклов

Прежде чем применять данное преобразование, необходимо проверить условие эквивалентности [28], а именно:
Пусть дана матрица T – матрица преобразования и гнездо циклов (*).
Построим вектор направления зависимости для вхождений гнезда циклов (*).
Тогда преобразование будет эквивалентно, если выполнены условия:

1. T – Невырожденная матрица.
2. Зависимости типа *in-out, out-in, out-out* – не изменятся, если преобразованные векторы направления зависимостей лексикографически положительны, т.е. $T * d_i > \overline{0}(\forall d_i)$, где d_i - векторы направления зависимостей.

3.7. Алгоритм Фурье-Моцкина

Границы нового гнезда циклов могут быть найдены из матричного неравенства $A' * J \le \beta$ с помощью алгоритма Фурье-Моцкина. Этот алгоритм уже неоднократно применялся в этих же целей в работах [18, 30].

Алгоритм Фурье-Моцкина требует *n-1*- го выполнения цикла, где n – число элементов вектора *J*. На каждой итерации получаем границы выпуклого многогранника по одной из размерностей. Порядок, в котором вычисляются границы, имеет значения. Они должны вычисляться от самого внутреннего к самому внешнему циклу.

Алгоритм Фурье-Моцкина описан в работе [7]. Там же представлены его временные характеристики.

3.8. Приведение матрицы преобразования к её Эрмитовой нормальной форме

Теорема 9. Пусть $A = [a_{i,j}]_{i=1}^{m}, {}_{j=1}^{n} \in M_{m \times n}(Z)$ (m<n) , ранг матрицы равен *m*. Тогда найдется унимодулярная матрица $V = [v_{i,j}]_{i=1}^{n}, {}_{j=1}^{n} \in M_{n \times n}(Z)$ такая, что матрица H = A*V имеет следующий вид:

$$\begin{pmatrix} h_{11} & 0 & . & . & . & 0 & \dots & 0 \\ h_{21} & h_{22} & . & . & . & 0 & \dots & 0 \\ & & . & & & & & \\ & & & . & & & & \\ h_{m-11} & h_{m-12} & & & . & 0 & \dots & 0 \\ h_{m1} & h_{m2} & & & & h_{mm} & \dots & 0 \end{pmatrix}$$

где $h_{i,i} \neq 0$; и все $h_{i,j}$ - целые числа.

Матрица H называется Эрмитовой нормальной формой матрицы A [9].

При этом выполняются следующие условия:

1. Максимальные значения каждой строки матрицы H принадлежат главной диагонали.
2. Также потребуем, для удобства использования, чтобы все элементы матрицы были не меньше нуля. К этому всегда можно прийти последовательно применяя элементарные преобразования над столбцами

матрицы.

Теорема 9 [37, 42]**.** Пусть $T,\ C \in M_{n\times n}(Z)$, где C - унимодулярная матрица и $H = T*C$ – Эрмитова нормальная форма матрицы T.
Тогда $Im\ (T*C) = Im\ (H) = Im\ (T)$.

Таким образом, учитывая предыдущую теорему, для формирования границ циклов нового гнезда, вместо матрицы преобразования T можно рассматривать её Эрмитову нормальную форму.

Пример 18. В данном примере показывается, в частном случае, как находить точные границы циклов для нового гнезда, после действия неунимодулярной матрицы на исходное гнездо.

На рис. 3 показана часть пространства итераций представленного на рис. 2b

H = $\begin{pmatrix} h_{11} & 0 \\ h_{21} & h_{22} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 2 & 3 \end{pmatrix}$ - Эрмитова нормальная форма матрицы

преобразования T_2 из примера 17.

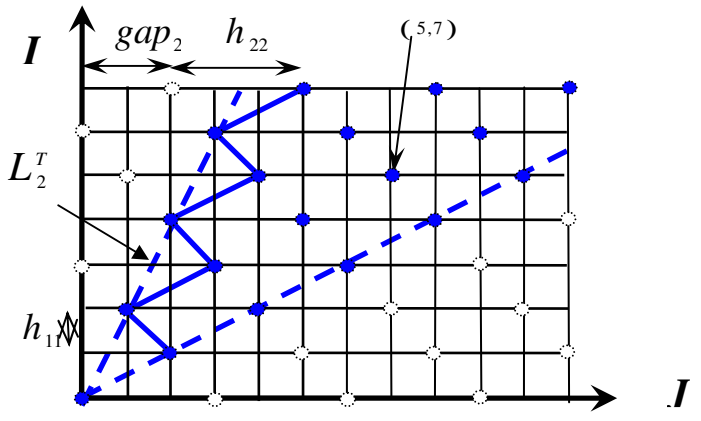


Рис. 3 Часть пространства итераций представленного на рис. 2b и параметры, характеризующие новое гнездо циклов.

Здесь:

L_2^T - Функция от счетчиков внешних циклов, полученная с помощью алгоритма Фурье-Моцкина;

gap_2 - Дополнительная функция от счётчиков внешних циклов.

Белые и синие точки на рис. 3 имеют целочисленные прообразы. Таким образом, если $J = (j_1, j_2)$ – одна из таких точек, т.е. $H^{-1} * J = I \in Z^2$ то координаты этой точки имеют вид:

$$j_1 = h_{11} * c_1$$

$$j_2 = h_{22} * c_2 + gap_2$$

где c_1, c_2 - целые числа, которые находятся с помощью алгоритма Фурье-Моцкина, а функция gap_2 имеет вид:

$$gap_2 = \left(\frac{h_{21}}{h_{11}} * j_1 \right) \bmod h_{22}$$

Для примера, рассмотрим одну из таких точек (5, 7), тогда можно написать:

$$(5, 7) = (5 * h_{11}, 2 * h_{22} + (2 * 5) \bmod 3) = (5 * 1, 2 * 3 + 1)$$

В n – мерном случае, каждая координата j_i точки $J = (j_1, j_2, \ldots, j_n)$ есть сумма двух элементов. Первый элемент – это произведение h_{ii} и некоторого целого числа, найденного с помощью алгоритма Фурье-Моцкина. Второй элемент, именуемый как gap_i - это дополнительная функция, которая включает в себя координаты от j_1 до j_{i-1}, $(i=1..n)$ [28].

Теорема 10. Пусть дано гнездо циклов (10), матрица преобразования T и пусть выполняются следующие условия:

1. Шаг k – го преобразованного цикла равен h_{kk}, где $1 \le k \le n$, n – размерность гнезда циклов (10), h_{kk} - элемент матрицы H, которая является Эрмитовой нормальной формой матрицы T.

2. Начальное значение для k – го счётчика цикла преобразованного гнезда имеет вид: $j_k = \left\lceil \frac{L_k^T - gap_k}{h_{kk}} \right\rceil * h_{kk} + gap_k$, где L_k^T - нижней границей выпуклого многогранника преобразованного пространства итераций вдоль координаты k и $gap_k = \left(\sum_{r=1}^{k-1} h_{kr} * i_r \right) \bmod h_{kk} = \left(\sum_{r=1}^{k-1} h_{kr} * \sum_{l=1}^{r} h'_{rl} * j_l \right) \bmod h_{kk}$, где h'_{ij} - элементы матрицы H^{-1} и j_l - $l^{й}$ счётчик преобразованного гнезда циклов $1 \le l \le n$.
3. Индексные выражения в теле гнезда циклов (10), примут вид: $f\left(T^{-1} * (j_1, j_2, \ldots, j_n)\right)$.
4. Условие эквивалентности преобразования. Данное условие описано в разделе «Условия эквивалентности неунимодулярного преобразования циклов».

Тогда гнездо циклов (10) можно заменить на равносильное ему гнездо:

For $j_1 = \left\lceil \frac{L_1^T}{h_{11}} \right\rceil * h_{11}; U_1^T;\quad h_{11}$ do

$gap_2 = (h_{21} * h'_{11} * j_1) \bmod h_{22}$

For $j_2 = \left\lceil \frac{L_2^T - gap_2}{h_{22}} \right\rceil * h_{22} + gap_2; U_2^T;\quad h_{22}$ do (**)

. . .

$gap_k = \left(\sum_{r=1}^{k-1} h_{kr} * \sum_{l=1}^{r} h'_{rl} * j_l \right) \bmod h_{kk}$

For $j_k = \left\lceil \frac{L_k^T - gap_k}{h_{kk}} \right\rceil * h_{kk} + gap_k;\quad U_k^T;\quad h_{kk}$ do

. . .

LOOPBODY [$f\left(T^{-1} * (j_1, j_2, \ldots, j_n)\right)$]

EndFor

EndFor

. . .

EndFor

Где L_k^T и U_k^T - границы k – го цикла преобразованного гнезда, полученные с помощью алгоритма Фурье – Моцкина.

Доказательство см. [28].

Замечание. Необходимо заметить, что верхняя граница преобразованного гнезда циклов остаётся без изменения, т.е. имеет вид U_k^T, Это следует из неравенства:

$$\left\lceil \frac{L_k^T - gap_k}{h_{kk}} \right\rceil * h_{kk} - \left\lfloor \frac{L_k^T - gap_k}{h_{kk}} \right\rfloor * h_{kk} - \left(L_k^T - gap_k\right) * \bmod h_{kk} < h_{kk}.$$

Следствие. Если все h_{kk}, $1 \le k \le n$ равны единицы, то k – й цикл

$$\textbf{For } j_k = \left\lceil \frac{L_k^T - gap_k}{h_{kk}} \right\rceil * h_{kk} + gap_k ; U_k^T ; \textbf{ step = } h_{kk} \textbf{ do}$$

примет вид:

$$\textbf{For } j_k = L_k^T ;\ U_k^T \textbf{ do}$$

Таким образом, если матрица преобразования унимодулярная, то её Эрмитова нормальная форма является единичной матрицей. Тем самым данный метод даёт те же результаты, что и подобные методы, решающие эту задачу для унимодулярных матриц.

Пример 18. Пример использования неунимодулярного преобразования циклов для векторизации.

Рассмотрим гнездо циклов (11) и матрицу преобразования T.

For I = 0, 4 do

```
For J = 0, 3 do                    (11)
  For K = 0, 3 do
    U (J + 2, K + 2) = (U (J + 1, K + 2) + U (J + 2, K + 1)) / 2
    EndFor
  EndFor
EndFor
```

Матрица преобразования

$$T = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad |T| = -1$$

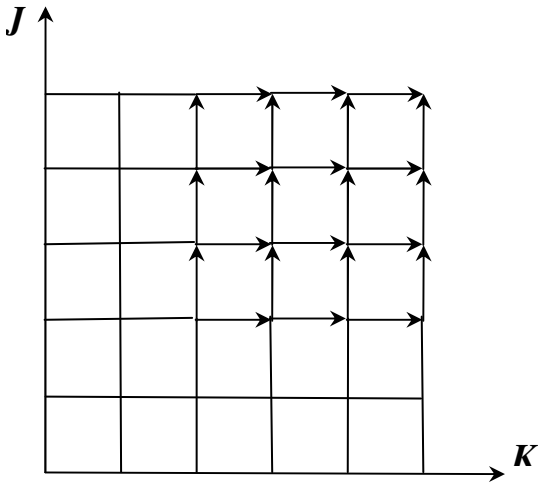


Рис. 6а Пространство итераций гнезда (11) для циклов со счётчиками **k** и **j.** Стрелками показаны направления векторов зависимостей

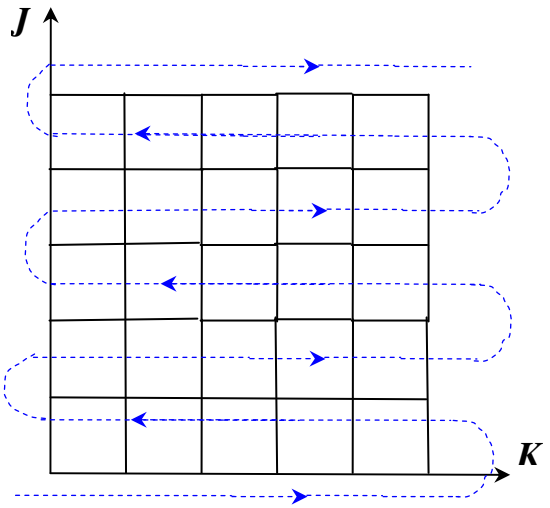


Рис. 6б Порядок выполнения итераций в гнезде (11) для циклов со счетчиками **k** и **j**

На рис. 6а представлен граф программы (11). Стрелками показаны направления векторов зависимостей (0, 0, 1) и (0, 1, 0). Данный цикл не векторизуемый, так как оператор присваивания в его теле рекуррентен (нельзя, например, прочитать вторую итерацию цикла до того, как была выполнена первая). На рис. 6б пунктирной линией показано направление выполнения циклов со счетчиками j и k. После применения неунимодулярного преобразования к циклу (11), с матрицей преобразования *T* получим гнездо циклов следующего вида:

For i1 = 0; 14 do

Lt2 = max ($\left\lceil \frac{i_1 - 6}{2} \right\rceil$, 0);

Ut2 = min ($\left\lfloor \frac{i_1}{2} \right\rfloor$, 4);

For j1 = Lt2; Ut2 do

Lt3 = max (i1 – 2 * j1 – 3, 0);

Ut3 = min (3, i1 – 2 * j1); (12)

For k1 = Lt3; Ut3 do

i = j1;

j = i1 – 2 * j1 – k1;

k = k1;

```
        U (J + 2, K + 2) = (U (J + 1, K + 2) + U   (J + 2, K + 1)) / 2
      EndFor
    EndFor
EndFor
```

На рис. 6в показано направление выполнения циклов со счетчиками j и k, которое получено после применения неунимодулярного преобразования. Таким образом, выполнения предыдущих итераций не зависят от выполнения последующих, что даёт возможность векторизовать циклы по j и k. Эта задача впервые была рассмотрена Лампортом в работе [33].

С помощью этой задачи могут быть хорошо распараллелены сеточные методы решения дифференциальных и интегральных уравнений.

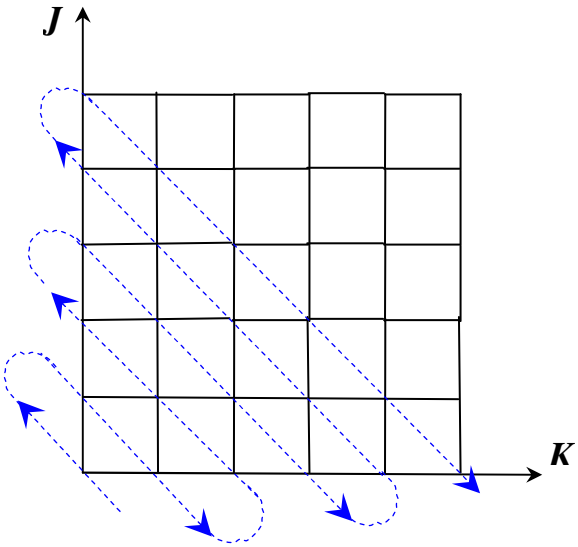


Рис. 6в Измененный порядок выполнения итераций в гнезде (11) для циклов со счетчиками **k** и **j**

Алгоритм неунимодулярного преобразования циклов

Для реализации преобразования циклов с помощью неунимодулярных матриц программно реализованы вспомогательные функции и библиотека матриц на языке программирования C++:

- Библиотека матриц с целыми элементами, со всеми необходимыми свойствами для неунимодулярного преобразования:
  1. Создать новую матрицу.
  2. Скопировать существующую матрицу
  3. Сложить две матрицы
  4. Найти разность двух матриц
  5. Перемножить две матрицы
  6. Умножить матрицу на вектор
  7. Умножить матрицу на число
  8. Извлечь подматрицу из заданной матрицы
  9. Сравнить две матрицы
  10. Элементарные столбцовые преобразования над матрицей
      - 10.1. Поменять местами два столбца
      - 10.2. Умножить столбец на ненулевое число
      - 10.3. Прибавить к одному столбцу другой столбец со скалярным множителем
  11. Найти минимальный элемент в заданной строке матрицы
  12. Получить заданную строку матрицы
  13. Получить количество строк, столбцов в матрице
  14. Получить любой минор матрицы

15. Получить любое алгебраическое дополнение матрицы
16. Получить определитель матрицы
17. Получить обратную матрицу к заданной
18. Получить транспонированную матрицу к заданной

- Распознавание применимости исходного гнезда циклов и матрицы преобразования.

Реализованы следующие проверки:

1. Являются ли границы циклов в гнезде константами. Для этого написана функция hasConstancyBounds(StmtFor* Loop).
2. Является ли исходное гнездо циклов тесным или нетесным. Для этого написана функция isPerfectlyNested (StmtFor* Loop)
3. Является ли шаг каждого цикла в исходном гнезде равным единице. Для этого написана функция hasStepEqualUnit (StmtFor* Loop).
4. Совпадает ли ранг матрицы преобразования и размерность исходного гнезда циклов. Для этого написана функция compatibilityMatrixAndLoop (StmtFor* Loop, const Matrix& T).
5. Является ли матрица преобразования невырожденной. Для этого написана функция isNonsingularMatrix (const Matrix& T).
6. Являются ли элементы матрица преобразования только целыми числами. Для этого написана функция matrixElementsIsInteger (const Matrix& T).
7. Является ли неунимодулярное преобразование циклов эквивалентным. Для этого написана функция isEquivalenceConversion (StmtFor* Loop, const Matrix& T), которая получает исходное гнездо циклов и матрицу преобразования. Для всех пар вхождений в теле цикла строятся векторы расстояния

зависимости d_i. После этого умножается каждый d_i на матрицу преобразования T и проверяется, что $T * d_i > \bar{0}$ лексикографически, $(\forall d_i)$. Если данное условие нарушается, то на экран выводится сообщение, что данное преобразование неэквивалентно.

- Реализация алгоритма Фурье-Моцкина

По исходному гнезду циклов формируется расширенная матрица

Пример 19. Гнездо циклов:

```
For i1 = 0, 4 do
   For i2 = 0, 6 do
     A [i1, i2] = A [i1 - 1, i2]
   EndFor
EndFor
```

Расширенная матрица исходного гнезда цикла

$$\left(\begin{array}{cc|c} 1 & 0 & 4 \\ 0 & 1 & 6 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{array}\right)$$

1. Полученная расширенная матрица преобразуется матрицей преобразования *T*.
   Для этого вычисляется обратная матрица к матрице преобразования T. После этого расширенная матрица перемножается с матрицей T^{-1}. Таким образом, получаем преобразованную расширенную матрицу.

2. К преобразованной расширенной матрице применяется алгоритм Фурье-Моцкина. Этот алгоритм реализован в соответствии с описанием в [7].

После выполнения алгоритма Фурье-Моцкина получаем выражения для границ нового гнезда циклов вида: max и min от счетчиков внешних циклов. Для самого внешнего цикла получаем целые константы.

- Алгоритм приведения матрицы преобразования к её Эрмитовой нормальной форме.

Этот алгоритм реализован в соответствии с описанием в [9]. Функция algorithmOfHermite (const Matrix& matrix), получает матрицу преобразования, и к ней применяются элементарные столбцовые преобразования. Алгоритм, описанный в [9], не гарантирует, что все элементы результирующей матрицы будут неотрицательными. Реализована функция nonnegativity (const Matrix& matrix), которая получает результирующую матрицу и с помощью столбцовых элементарных преобразований над ней приводит её к матрице с неотрицательными элементами.

4. Программная реализация в Открытой распараллеливающей системе

Открытая распараллеливающая система (ОРС) — разрабатывается студентами и аспирантами кафедры алгебры и дискретной математики РГУ. Эта система предназначена для автоматического распараллеливания программ с процедурных языков программирования (ФОРТРАН, Паскаль, Си) на параллельные суперкомпьютеры. Автоматически распараллеливающие и оптимизирующие преобразования являются центральным этапом обработки исходного программного кода в ОРС.

ОРС состоит из нескольких частей: парсера (разборщика текста программы), библиотеки преобразований и генератора кода. Парсер, используя грамматику входного языка, преобразует исходный текст программы во внутреннее представление. Во внутреннем представлении программа выглядит как совокупность древовидных структур, узлы которых представляют операторы, операции и идентификаторы языка, а дуги между ними представляют отношения принадлежности и вложенности. Для внутреннего представления характерно то, что его легко можно перевести обратно в исходный текст, а также над ним легко совершать такие редакторские операции как удаление, вставка, копирование и замена.

Библиотека преобразований - это группа модулей, каждый из которых представляет собой некоторое преобразование фрагмента программы во внутреннем представлении.

"Перестановка циклов" и "Преобразование циклов с помощью неунимодулярных матриц" являются частью библиотеки преобразований.

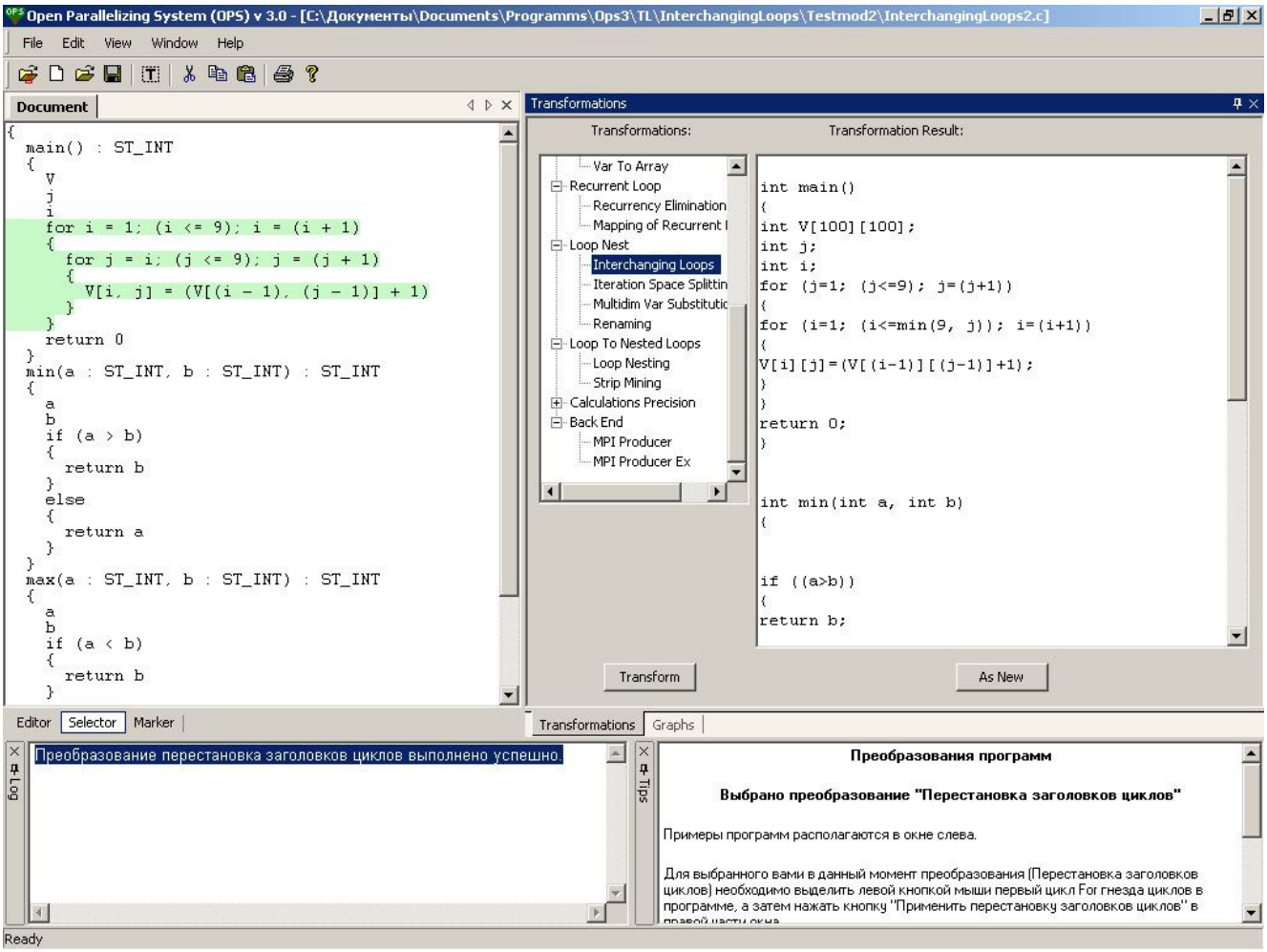


Рис. 4 Представлена визуализация оболочка OPC, в которой выполняется перестановка циклов в тесном двумерном гнезде с треугольным пространством итераций. Условия критерия перестановки циклов выполнены. Исходный цикл выделен зелёным цветом. Цикл после перестановки представлен в правом окошке.

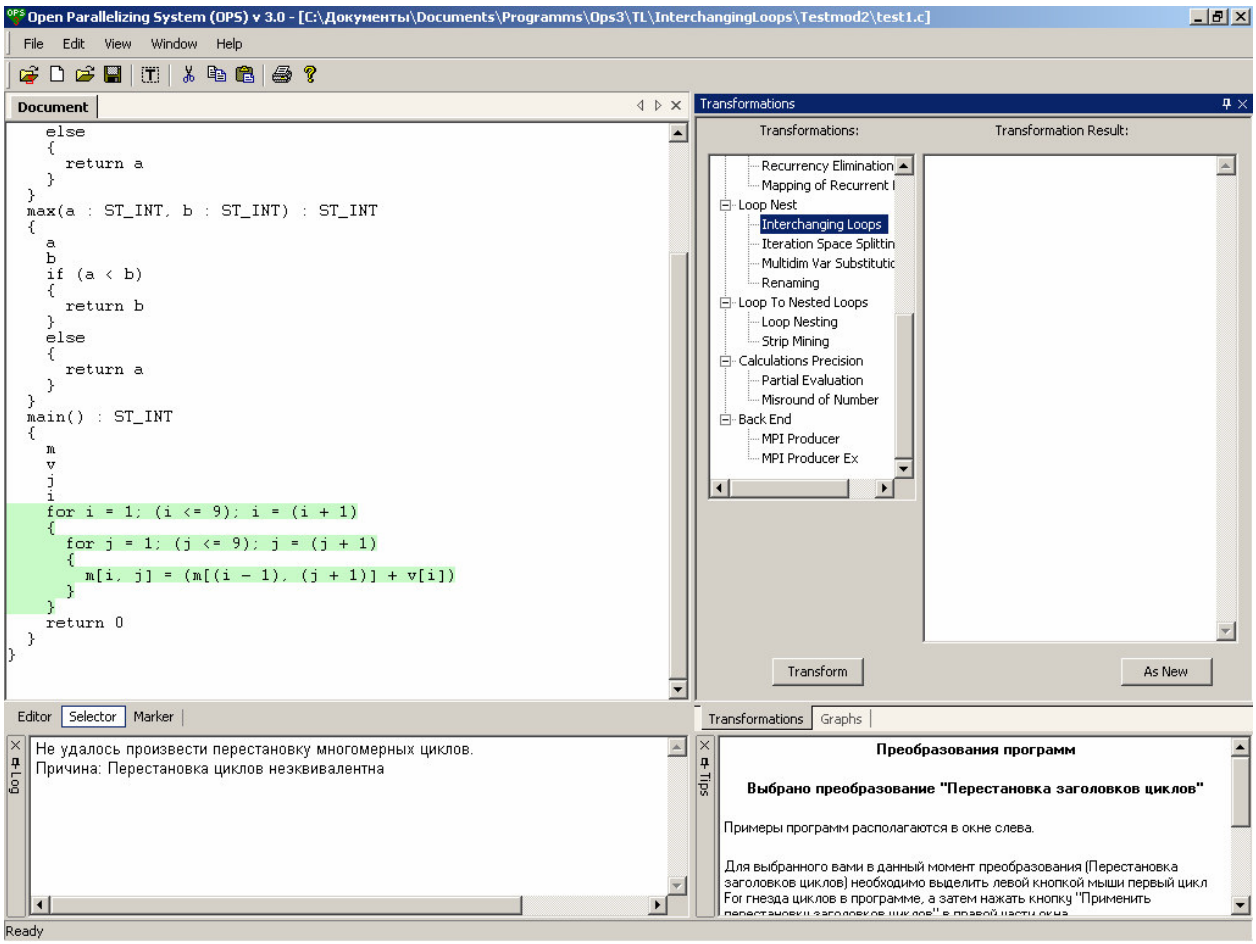


Рис. 5 Представлено тесное гнездо циклов. Условия критерия перестановки циклов нарушены. В теле данного цикла существует зависимость между вхождениями, имеющая такой вектор направления зависимости $\Psi = (\psi_1, \psi_2)$, что $\{<\} \subset \psi_1$ и $\{>\} \subset \psi_2$, который препятствует преобразованию.

Заключение

В магистерской работе рассмотрены оптимизирующие преобразования гнёзд циклов: перестановка циклов в тесном и нетесном гнезде и преобразование циклов с помощью неунимодулярных матриц.

Рассмотрены условия эквивалентности перестановки циклов в тесном гнезде. Сформулированы и доказаны необходимые и достаточные условия эквивалентности перестановки циклов в нетесном гнезде. Проведён сравнительный анализ, из которого стало ясно, что условий М. Вольфа для перестановки циклов в нетесном гнезде являются некорректными.

Приведены примеры использования перестановки циклов для многоконвейерных, векторных вычислительных систем. Показано использование перестановки циклов для оптимизации иерархии памяти.

Написан алгоритм реализации перестановки циклов в тесном гнезде.

Написан алгоритм реализации неунимодулярного преобразования, для которого были также реализованы алгоритмы: Фурье-Моцкина и Приведение матрицы к её эрмитовой нормальной форме.

Направления дальнейших исследований:

- Расширение класса решаемых задач для неунимодулярного преобразования.
- Формирование матриц преобразования, в зависимости от системных свойств и характеристик той или иной архитектуры.
- Сбор статистики перестановки циклов на реальных программах.

Приложение А

В таблице представлены некоторые преобразования циклов, которые можно получить с помощью унимодулярных матриц.

<table>
<tr><td colspan="2">ПЕРЕСТАНОВКА ЦИКЛОВ
(LOOP INTERCHANGE)
МАТРИЦА ПРЕОБРАЗОВАНИЯ
$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$</td></tr>
<tr><td>До преобразования

For I = 0, 5 do
  For J = 0, 5 do
  A(F(I, J))
  EndFor
EndFor</td><td>После преобразования

For I1 = 0, 5 do
  For J1 = 0, 5 do
  I = J1
  J = I1
  A(F(I, J))
  EndFor
EndFor</td></tr>
<tr><td colspan="2">СКАШИВАНИЕ ГНЕЗДА ЦИКЛОВ
(SKEWING)
МАТРИЦА ПРЕОБРАЗОВАНИЯ
$\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$</td></tr>
<tr><td>До преобразования

For I = 0, 5 do
  For J = 0, 5 do
  A(F(I, J))</td><td>После преобразования

For I1 = 0, 5 do
  For J1 = I1, I1 + 5 do
  I = I1</td></tr>
</table>

EndFor EndFor	J = J1 – I1 A(F(I, J)) EndFor EndFor
ИНВЕРСИЯ ЦИКЛА **(LOOP REVERSAL)** МАТРИЦА ПРЕОБРАЗОВАНИЯ (-1)	

До преобразования
 	
For I = 0, 5 do
For J = 0, 5 do
A(F(I, J))
EndFor
EndFor
	После преобразования
	
	For I1 = -5, 0 do
	I = -I1
	For J1 = 0, 5 do
	A(F(I, J))
	EndFor
	EndFor

Литература

1. Аллен Р., Кеннеди К. Автоматическая трансляция Фортран-программ в векторную форму. Векторизация программ: теория, методы, реализация. Сб. статей. Москва: Мир. 1991. С.77 - 140.
2. Воеводин В. В. Математические модели и методы в параллельных процессах. Москва: Наука. 1986. – 296 с.
3. Вольф М. Перестановка циклов. Векторизация программ: теория, методы, реализация. Сб. статей. Москва: Мир. 1991. С. 48 – 65.
4. Сб. статей. Векторизация программ: теория, методы, реализация. Москва: "Мир". 1991. 272 с.
5. Евстигнеев В.А., Касьянов В.Н. Оптимизирующие преобразования в распараллеливающих компиляторах// Программирование, 1996, № 6, с. 12-26.
6. Касьянов В.Н., Оптимизирующие преобразования программ/ М., «Наука», 1988 г., 336 с.
7. Схрейвер А.. Теория линейного и целочисленного программирования. Москва: Мир. 1991. Т. 1. С. 239 – 242.
8. Падуа Д., Вольф М. Оптимизация в компиляторах для суперкомпьютеров. Векторизация программ: теория, методы, реализация. Сб. статей. Москва: Мир. 1991. С. 7-47.
9. Ковалёв М. М. Дискретная оптимизация (целочисленное программирование). Мн., Изд-во БГУ, 1977. С. 16-18.
10. Штейнберг Б. Я. Математические методы распараллеливания рекуррентных циклов для суперкомпьютеров с параллельной памятью.// Ростов-на-Дону, Издательство Ростовского университета, 2004 г., 192 с.
11. Штейнберг Б. Я. Открытая распараллеливающая система // PACO’2001/ Труды международной конференции «Параллельные вычисления и задачи управления». М.: ИПУ РАН, 2001. С. 214-220.
12. Штейнберг Б. Я. Распараллеливание программ для суперкомпьютеров с параллельной памятью и Открытая распараллеливающая система.

Диссертация на соискание ученой степени доктора технических наук. РГУ, 2004.

13. Штейнберг Б.Я., Арутюнян О.Э., Бутов А.Э., Гуфан К.Ю., Морылев Р., Науменко С.А., Петренко В.В., Тузаев А., Черданцев Д.Н., Шилов М.В., Штейнберг Р.Б., Шульженко А.М. Обучающая распараллеливанию программа на основе ОРС.// Научно-методическая конференция «Современные информационные технологии в образовании: Южный федеральный округ», Ростов-на-Дону, 12-15 мая 2004 г., с. 248-250.
14. Штейнберг Б.Я., Бутов А.Э., Науменко С.А., Петренко В.В., Черданцев Д.Н., Штейнберг Р.Б., Шульженко А.М. Полуавтоматическое распараллеливание на основе ОРС.// Труды всероссийской научно-технической конференции «Параллельные вычисления в задачах математической физики», 21-25 июня 2004, г. Ростов-на-Дону, с. 186-194.
15. Штейнберг Б. Я., Макошенко Д. В., Черданцев Д. Н., Шульженко А. М. Внутреннее представление в Открытой распараллеливающей системе. //Искусственный интеллект. Научно-теоретический журнал. Институт проблем искусственного интеллекта НАНУ. Украина, Донецк, ДонДИШИ, «Наука и Освита», 2003, №4, с. 89-97.
16. Шульженко А. М. Определения информационных зависимостей в программе при автоматическом распараллеливании. Диссертация на соискание ученой степени магистра. РГУ, 2002.
17. Allen R., Kennedy K. Optimizing compilers for modern architectures. Morgan Kaufmann Publishers, An Imprint of Elsevier. 2002. – 790 p.
18. Ancourt C., Irigoin F. Scanning polyhedra with DO loops. // In ACM/SIGPLAN Symp. on Principles and Practice of Parallel Programming, pages 39--50, ACM, June 1991.
19. Banerjee U., Chen S. C., Kuck D., Towle R. Time and Parallel Processor Bounds for Fortran-like Loops// IEEE Trans. on Computers. 1979. C-28. No.9. P.660-670.

20. Banerjee U. Unimodular transformations of double loops. Advances in Language and Compilers for Parallel Processing. MIT Press. 1991. C. 192-219.
21. Chamski Z. Fast and efficient generation of loop bounds. // Proceedings of ParCo '93, Elsevier Science Publishers (North Holland).
22. Chamski Z. Nested Loop Sequences: Towards Efficient Loop Structures in Automatic Parallelization. //HICSS 1994: Maui, Hawaii, USA - Volume 2. P 14-22.
23. Collard J.-F., Feautrier P. Automatic Generation of Data Parallel Code.//Proceedings of the Fourth International Workshop on Compilers for Parallel Computers. Delft, The Netherlands, Dec. 1993, pages 321-332.
24. Collard J.-F., Feautrier P., Risset T. Construction of DO Loops from Systems of Affine Constraints. Research Report 93-15, Ecole Normale Superieure de Lyon, Lyon, France, May 1993.
25. Fernandez A. Systematic transformation of systolic algorithms for programming multicomputers. PhD thesis, Universidad Politecnica de Cataluna, Barcelona, Spain, 1992.
26. Feautrier P. Parametric integer programming// Operationnelle/Operations Research, 22(3):243--268, September 1988.
27. Feautrier P. Dataflow analysis of scalar and array references// International Journal of Parallel Programming, 20(1):23--52, February 1991.
28. Fernandez A., Llaberia J.M., Valero-García M. Loop Transformation Using Non-unimodular Matrices. IEEE Transactions on Parallel and Distributed Systems. Vol. 6. No. 8, Aug. 1995.
29. Kelly W., Pugh W. A framework for unifying reordering transformations.// Technical Report: CS-TR-2995. University of Maryland at College Park, MD, USA. 1993.
30. Khun R.H. Optimization and interconnection complexity for: Parallel processors, single-stage networks and decision trees. PhD thesis, Dept. of Computer Science, University of Illinois at Urbana-Champaign, Feb 1980.

31. Kumar K. G., Kulkarni D., Basu A. Generalized Unimodular Loop Transformations for Distributed Memory Multiprocessors. Center for Development of Advanced Computing. International Conference of Parallel Processing, 1991.
32. Kumar K. G., Kulkarni D., Basu A. Deriving Good Transformation for Mapping Nested Loops on Hierarchical Parallel Machines in Polynomial Time. Center for Development of Advanced Computing. International Conference on Supercomputing 1992, Washington.
33. Lamport L. The parallel execution of DO loops// Commun. ACM. 1974. Vol.17. No.2. P.83-93.
34. Li W. Pingali K. Access normalization: Loop restructuring for NUMA compilers. Tech. Report TR92-1278, Dept. of Computer Science, Cornell University, Apr 1992.
35. Li W. Pingali K. A singular loop transformation framework based on non-singular matrices. Tech. Report TR92-1294, Dept. of Computer Science, Cornell University, Apr 1992.
36. Maydan D. E., Amarasinghe S. P., Lam M. S.. Data dependence and data-flow analysis of arrays// In 5th Workshop on Languages and Compilers for Parallel Computing, New Haven, CT, August 1992.
37. Nemhauser G.L., Wosley L.A. Integer and Combinatorial Optimization. Wiley-Interscience Series in Discrete Mathematics and Optimization. 1988.
38. Pugh W. Definitions of Dependence Distance// Letters on Programming Languages and Systems, September 1993.
39. Ramanujar J. Compiler-Time techniques for parallel execution of loops on distributed memory multiprocessors. PhD thesis, Ohio State University, 1990.
40. Ramanujar J. Non-unimodular Transformation of Nested Loops. Proc. Supercomputers'92. 1992. C. 214-223.
41. Ramanujar J. Beyong unimodular transformations. (To appear).
42. Schrijver A. Theory of linear and integral programming. Wiley-Interscience Series in Discrete Mathematics and Optimization. 1988.

43. Suárez A., Llaberia J.M., Fernandez A. Scheduling partitioning in systolic algorithms. IEEE International Conference Application-Specific Array Processors (ASAP'92), CS Press, 1992. C. 619-663.
44. Suárez A. Scheduling partitions on systolic algorithms. PhD thesis, Universidad Politecnica de Cataluna, Barcelona, Spain, 1993.
45. Wolfe M. Optimizing Super compilers for supercomputers. Research Monographs in Parallel and Distributed Computing. London Pitman Publishing, 1989.
46. Wolfe M., Banerjee U. Data Dependence and Its Application to Parallel Processing// International Journal of Parallel Programming. 1987. Vol.16. No.2. P.137-178.
47. Wolfe M. E., Lam M. S. A loop transformation theory and an algorithm to maximize parallelism// IEEE Transactions on Parallel and Distributed Systems, July 1991.
48. http://citforum.iubip.ru
49. http://www.cluster.bsu.by
50. http://security.ultrabait.net