

УДК 004.424.22.004.312.46

 10.25209/2079-3316-2024-15-1-63-94

Обоснование методов ускорения гнёзд циклов итерационного типа

Елена Анатольевна **Метелица**[✉]

Южный Федеральный Университет, Ростов-на-Дону, Россия

[✉]metelica@sfedu.ru

Аннотация. Рассматривается ускорение итерационных алгоритмов, которые встречаются при решении задач математической физики, математического моделирования, обработки изображений и других. В программной реализации таких алгоритмов лежат гнёзда циклов (участки программы, состоящие из вложенных циклов). Такие гнёзда циклов ускоряются при помощи комбинации оптимизирующих преобразований, включающих тайлинг, метод гиперплоскостей и распараллеливание на общую память. Обосновывается эквивалентность комбинации используемых преобразований программ.

Предлагается и обосновывается метод изменения порядка обхода тайла. Метод даёт ускорение за счёт увеличения количества чтений данных из регистров, вместо чтений из более медленной памяти. С учётом этого метода получена формула вычисления оптимальных размеров тайлов.

Представленной в статье цепочкой преобразований достигается ускорение в 1.4 раза больше, чем в известном алгоритме оптимизации, реализованном в системе PLUTO. Приводятся численные эксперименты, которые в некоторых случаях на процессоре с 8 ядрами демонстрируют ускорение относительно исходных последовательных программ более чем на порядок. Результаты статьи могут использоваться для ручной и автоматизированной оптимизации программ.

Ключевые слова и фразы: тайлинг, метод гиперплоскостей, распараллеливание, общая память, гнёзда циклов итерационного типа

Благодарности: Автор благодарен д.т.н. Б.Я. Штейнбергу за руководство работой и Ар.В. Климову за внимание и интерес к работе.

Для цитирования: Метелица Е. А. *Обоснование методов ускорения гнёзд циклов итерационного типа* // Программные системы: теория и приложения. 2024. Т. 15. № 1(60). С. 63–94. https://psta.psiras.ru/read/psta2024_1_63-94.pdf

Введение

Рассматривается ускорение алгоритмов, которые встречаются при решении задач математической физики, математического моделирования, обработки изображений и других. В программной реализации рассматриваемых алгоритмов лежат гнёзда циклов (участки программы, состоящие из вложенных циклов) итерационного типа. Отдельные преобразования, которые используются для оптимизации таких гнёзд циклов описаны во многих работах. В частности, теория оптимизации гнёзд циклов представлена в работах М. Lam [1], М. Wolf [1, 2], У. Banerjee [2], У. Bondhugula [5], Л. Lamport [4] и основывается на преобразованиях пространства итераций в форме полиэдра. Автоматизация некоторых ключевых преобразований, используемых в данной работе, реализована в некоторых системах, таких как Polly LLVM¹, PLUTO², PolyMage [5] и др. В работах У. Bondhugula приводится подход к оптимизации гнёзд циклов итерационного типа. Этот подход автоматизирован в системе PLUTO. В статье [14] рассматривается подход к распараллеливанию гнёзд циклов итерационного типа с использованием алгоритма, основанного на преобразованиях «скошенный тайлинг» и «метод гиперплоскостей».

Данная работа рассматривает представленный в [10], [14] алгоритм распараллеливания гнёзд циклов итерационного типа для вычислительных систем с общей памятью. Расширяет его дополнительными преобразованиями (перестановка циклов внутри тайла, линейаризация [16], вынос инвариантных выражений) и даёт теоретическое обоснование его эквивалентности.

Описывается метод изменения обхода точек тайла, повышающий временную локальность данных. Изменение обхода точек тайла достигается за счёт перестановки циклов. Обосновывается целесообразность использования этого метода. Приводится модель вычисления оптимальных размеров тайлов, которая подтверждается численными экспериментами.

Численные эксперименты выполнения программ, преобразованных при помощи алгоритма, описанного в данной работе, демонстрируют ускорение относительно их исходных последовательных версий. Например, ускорение алгоритма Гаусса-Зейделя для численного решения обобщенной задачи Дирихле уравнения Лапласа составляет 16.32 раза. Достигнутый результат превосходит в 1.4 раза ускорение, полученное при помощи

¹Polly - Polyhedral optimizations for LLVM.  <https://polly.llvm.org/>.

²PLUTO - An automatic parallelizer and locality optimizer for affine loop nests.

 <https://pluto-compiler.sourceforge.net/>.

оптимизирующей системы PLUTO. Ускорение достигается посредством дополнительных преобразований, повышающих временную локальность данных. Также приводится ускорение, достигаемое описанным алгоритмом с использованием дополнительных преобразований.

Результаты статьи могут использоваться при разработке быстрых программ и оптимизирующих компиляторов.

1. Преобразования программ для ускорения целевых алгоритмов

Приведём некоторые определения для лучшего понимания результатов работы. Будут использоваться описания известных типов зависимости: истинная, антитезис, циклически порожденная зависимость и другие термины теории преобразований программ [12].

Вхождением переменной будем называть всякое появление переменной в тексте программы вместе с тем местом (строка и позиция имени переменной в строке) в программе, в котором эта переменная появилась. Всякому вхождению (при конкретном значении индексного выражения для массивов) соответствует обращение к некоторой ячейке памяти. Например, на рисунке 1 изображен текст программы, в которой присутствуют три вхождения массива u .

Генератор (out , $output$) – вхождение, при котором происходит запись в ячейку памяти.

Использованиями (in , $input$) называются остальные вхождения переменных.

Граф информационных зависимостей – это ориентированный граф, вершины которого соответствуют вхождениям переменных, а дуга (u, v) соединяет пару вершин, если эти вхождения порождают информационную зависимость (обращаются к одной и той же ячейке памяти), причем вхождение u раньше обращается к общей ячейке памяти, чем v , и хотя бы одно из этих вхождений является генератором.

Информационная зависимость между вхождениями называется *циклически независимой* ($loop\ independent\ dependence$), если эти вхождения обращаются к одной и той же ячейке памяти на одной и той же итерации цикла. Иначе зависимость называется *циклически порожденной* ($loop\ carried\ dependence$), а цикл называется *создающим* такую зависимость.

Носителем информационной зависимости называют цикл, создающий эту зависимость, которая является циклически порожденной. Носители нумеруются от внешнего цикла к внутреннему, начиная с нуля. Для одной

циклически порождённой зависимости может быть несколько циклов, которые её создают. Обозначим множество носителей - *carriers*.

Рассмотрим граф информационных зависимостей (рисунок 1) гнезда циклов. Для каждой дуги графа вычислим множество зависимостей:

$$\begin{aligned} \text{carriers}(u_{i,j}, u_{i,j}) &= (0), & \text{carriers}(u_{i,j}, u_{i-2,j-1}) &= (0, 1), \\ \text{carriers}(u_{i-2,j-1}, u_{i,j}) &= (0), & \text{carriers}(u_{i,j}, u_{i+1,j-1}) &= (0), \\ \text{carriers}(u_{i+1,j-1}, u_{i,j}) &= (0, 1). \end{aligned}$$

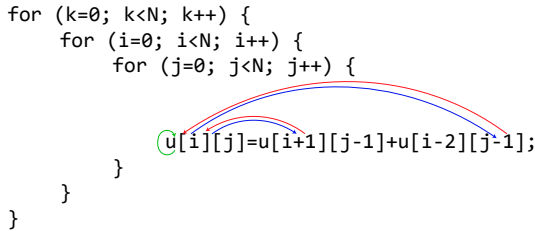


Рисунок 1. Граф информационных зависимостей трехмерного гнезда циклов

Рассмотрим гнездо из $n + 1$ вложенных друг в друга циклов. Прономеруем операторы циклов в этом гнезде в соответствии с порядком вложенности, начиная с самого внешнего цикла. Обозначим счетчик j -го цикла - I_j , где j от 0 до n . Множество значений, которые может принимать вектор счетчиков циклов $I = (I_0, I_1, \dots, I_n)$ в указанном гнезде, называется пространством итераций данного гнезда.

Если сделать раскрутку всех циклов гнезда, то получим код без циклов, состоящий из множества блоков, каждый из которых является копией тела исходного гнезда циклов. Каждая такая копия тела гнезда циклов соответствует набору значений счетчиков циклов гнезда.

Будем обозначать копию вхождения v в раскрутке исходного гнезда циклов при значениях счетчиков $I' = (I'_0, I'_1, \dots, I'_n)$ следующим образом: $v(I'_0, I'_1, \dots, I'_n)$ или $v(I')$ и называть представителем данного вхождения.

Для анализа и преобразований многомерных циклов используются решетчатые графы [15]. Приведём определение элементарного решетчатого графа из статьи [11].

Пусть X - это m -мерный массив, тогда будем рассматривать вхождение $X_{F(I)}$, где $F(I)$ - отображение пространства итераций в n -мерное целочисленное пространство индексов массива X , а I - точка пространства итераций. Далее отображение F будем рассматривать аффинным.

Например, для вхождения $X_{i_1-i_2, i_3+2}$ при размерности пространства итераций $n = 3$ и размерности массива X $m = 2$, отображение будет иметь вид: $F(i_1, i_2, i_3) = (i_1 - i_2, i_3 + 2)$.

Пусть в гнезде циклов имеется пара зависимых вхождений $X_{F(I)}$ и $X_{G(J)}$, и этой паре соответствует некоторая дуга графа информационных зависимостей $(X_{F(I)}, X_{G(J)})$ (листинг 1). Для этой дуги определим элементарный решетчатый граф (F, G) . Вершины решетчатого графа – точки пространства итераций гнезда циклов. Пусть (I, J) – пара точек пространства итераций и $I < J$ (лексикографический порядок, то есть I раньше J). Дуга (I, J) принадлежит графу, если $F(I) = G(J)$ и для любой точки пространства итераций K , для которой $K < J$ и $F(K) = G(J)$, выполняется $K \leq I$. Иными словами, вершина I является лексикографическим максимумом множества всех таких точек K , для которых $F(K) = G(J)$ и $K < J$. Если для любой переменной в теле гнезда циклов есть не более одного генератора, то существует взаимно однозначное соответствие между дугами графа (F, G) и не ложными дугами графа информационных зависимостей раскрутки всех циклов гнезда [11].

Листинг 1. Гнездо циклов с вхождениями массива X

```

for (int I = a; I < b; I++) {
    X[F(I)] = . . .
    . . .
    . . . = X[G(I)]
}
    
```

В каждую точку пространства итераций входит не более одной дуги элементарного решетчатого графа, соответствующей дуге $(X[F(I)], X[G(J)])$ графа информационных зависимостей.

Далее будем рассматривать решетчатый граф программы как объединение элементарных решетчатых графов.

Если в гнезде n циклов, то пространство итераций, т. е. множество наборов значений векторов счетчиков циклов после раскрутки образует подмножество целочисленной решетки n -мерного пространства. Поэтому графы информационных зависимостей между копиями тела гнезда циклов, естественно, называть решетчатыми [11].

Обозначим $u(i_1, i_2, \dots, i_d)$ представитель вхождения u в теле гнезда циклов с координатами (i_1, i_2, \dots, i_d) . Также $u(i_1, i_2, \dots, i_d)$ будем обозначать семейство вершин решетчатого графа, соответствующих вхождению u .

ПРИМЕР 1. Рассмотрим гнездо циклов.

```
for (i=1; i<=3; i=i+1)
  for (j=1; j<=3; j=j+1)
    a[j] = a[j+1];
```

Для этого гнезда циклов вхождению a_{j+1} соответствуют 9 представителей в раскрутке обоих циклов этого гнезда, каждой из которых соответствует точка пространства итераций решетчатого графа. Этому вхождению соответствует семейство представителей $a(i, j)$. При этом, например, $a(2, 3) = \langle a_4 \rangle$, $a(3, 1) = \langle a_2 \rangle$.

Результат полной раскрутки гнезда циклов

```
// i = 1
  a[1]=a[2]; // j = 1
  a[2]=a[3]; // j = 2
  a[3]=a[4]; // j = 3
// i = 2
  a[1]=a[2]; // j = 1
  a[2]=a[3]; // j = 2
  a[3]=a[4]; // j = 3
// i = 3
  a[1]=a[2]; // j = 1
  a[2]=a[3]; // j = 2
  a[3]=a[4]; // j = 3
```

Решетчатый граф, построенный ОРС для примера 1, показан на рисунке 2.

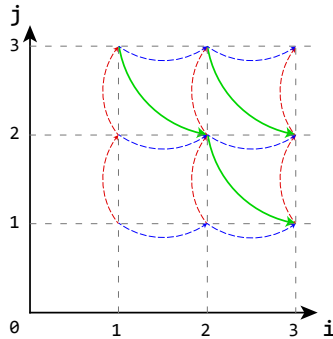


РИСУНОК 2. Решетчатый граф примера 1. Красная дуга - дуга антизависимости, зелёная - истинной зависимости, синяя - выходной зависимости

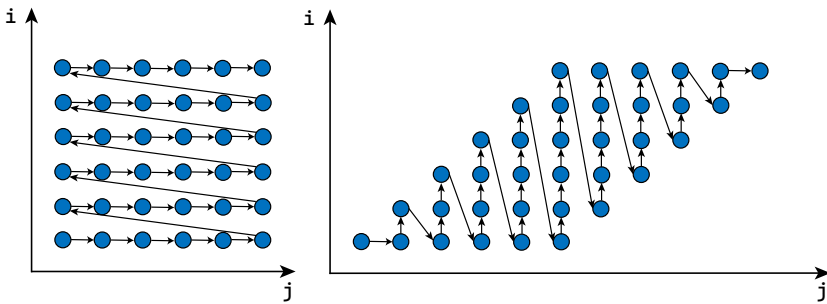
Скашивание (Loop Skewing) [7] – преобразование, которое изменяет пространство итераций гнезда циклов: вектор счетчиков нового гнезда циклов I' равен вектору счетчиков исходного гнезда I умноженного на нижнетреугольную матрицу skew с единицами на главной диагонали: $I' = \text{skew} \times I$. Число f , которое находится в строке m и в столбце k ($k < m$) будем называть параметром скашивания цикла номер k относительно цикла под номером m .

Матрица skew преобразования «скашивание циклов» для двумерного гнезда циклов имеет вид:

$$\text{skew} = \begin{pmatrix} 1 & 0 \\ f & 1 \end{pmatrix}.$$

Преобразование «скашивание циклов» (Loop Skewing) является эквивалентным.

На рисунке 3 представлено изменённое пространство итераций двумерного гнезда циклов после скашивания.



(а) до скашивания

(б) после скашивания при $f = 1$

РИСУНОК 3. Последовательность выполнения итераций двумерного гнезда циклов

Понятие *вектор расстояния (distance vector)* [6] относится к дуге графа информационных зависимостей. Вектор расстояния зависимости используется для определения параметров скашивания гнезда циклов для последующего корректного выполнения прямоугольного тайлинга.

Приведём определение вектора расстояний (distance vector) дуги информационной зависимости.

Пусть в графе информационных зависимостей гнезда из n циклов есть дуга (u, v) . Тогда существует такая пара представителей $u(i'_1, i'_2, \dots, i'_n)$, $v(i''_1, i''_2, \dots, i''_n)$ для которых есть дуга в решетчатом графе. Заметим, что таких пар представителей может быть много. Рассмотрим разность

векторов счетчиков циклов $(i''_1 - i'_1, i''_2 - i'_2, \dots, i''_n - i'_n)$. Если значение этой разности одинаково для всех таких дуг $(u(i'_1, i'_2, \dots, i'_n), v(i''_1, i''_2, \dots, i''_n))$ соответствующих (u, v) , то будем называть эту разность *вектором расстояния* (*distance vector*) дуги информационной зависимости (u, v) .

Гнездо циклов итерационного типа – это такое гнездо циклов [13], которое обладает следующими свойствами. Внешний цикл отвечает за обход итераций, а внутренние за расчёт шаблона вычислений, который повторяется на итерациях внешнего гнезда. В данной работе рассматриваются такие гнезда циклов итерационного типа, которые являются тесными и в тело которых входят только операторы присваивания, безындексные переменные, константы, массивы в стиле языка С; массивы имеют линейные индексные выражения вида $i + \text{целочисленная константа}$, где i счетчик одного из циклов. Более точно, в теле итерационного гнезда циклов ($\text{LoopBody}(i_1, i_2, \dots, i_n)$) могут присутствовать генераторы вида $a_{i_1 - c_1, i_2 - c_2, \dots, i_n - c_n}$, (где c_1, c_2, \dots, c_n – целочисленные константы), в которых последовательность индексов массива $(i_1 - c_1, i_2 - c_2, \dots, i_n - c_n)$ соответствует последовательности счетчиков гнезда циклов (i_1, i_2, \dots, i_n) . Тогда использования этой же переменной u в правой части оператора присваивания должны иметь вид: $a_{i_1 - d_1, i_2 - d_2, \dots, i_n - d_n}$, где d_1, d_2, \dots, d_n – целочисленные константы. Общий вид такого гнезда циклов представлен в листинге 2. Из-за наличия информационных зависимостей ни один цикл данного гнезда не может выполняться параллельно. Поэтому перед распараллеливанием надо выполнить преобразование.

Листинг 2. Гнездо циклов итерационного типа

```

for (int i0 = 0; i0 < itera; i0++)
  for (int i1 = a1; i1 < b1; i1++)
    for (int i2 = a2; i2 < b2; i2++)
      ...
      for (int in = an; in < bn; in++) {
        LoopBody(i1, i2, ..., in);
      }

```

Сформулируем понятие вектора расстояния для информационных зависимостей в гнезде циклов итерационного типа. В случае гнезда итерационного типа массивы, генераторы которых входят в тело гнезда, имеют размерность на единицу меньше, чем количество циклов гнезда.

Пусть размерность гнезда циклов итерационного типа равна $n + 1$, u и v – вхождения n -мерного массива a , образующие дугу информационной зависимости $(u, v) = (a_{i_1 - c_1, i_2 - c_2, \dots, i_n - c_n}, a_{i_1 - d_1, i_2 - d_2, \dots, i_n - d_n})$. Счетчик

внешнего цикла (имеющего порядковый номер 0), вычисляющего итерации алгоритма, не входит в индексные выражения массивов (согласно определению гнезда циклов итерационного типа). Тогда существует такая пара представителей, для которых есть дуга в решетчатом графе $(u(i'_0, i'_1, i'_2, \dots, i'_n), v(i''_0, i''_1, i''_2, \dots, i''_n))$.

Рассмотрим вектор длины $n + 1$ разностей векторов счетчиков циклов $(i''_0 - i'_0, i''_1 - i'_1, i''_2 - i'_2, \dots, i''_n - i'_n)$. Все координаты, начиная с первой, не зависят от выбора пары представителей u, v . А нулевая координата, всегда неотрицательна (поскольку дуга информационной зависимости направлена от u к v). Тогда определим для гнезда циклов итерационного типа вектор расстояний дуги информационной зависимости (u, v) формулой

$$\begin{aligned} \text{dist}(a_{i_1-c_1, i_2-c_2, \dots, i_n-c_n}, a_{i_1-d_1, i_2-d_2, \dots, i_n-d_n}) \\ = (1, d_1 - c_1, d_2 - c_2, \dots, d_n - c_n). \end{aligned}$$

Рассмотрим пример 1. Для дуги истинной информационной зависимости (a_j, a_{j+1}) вектор расстояний $= (1, -1)$.

Скашивание меняет векторы расстояний информационных зависимостей: матрица преобразования «скашивание циклов» для примера 1 имеет вид

$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix},$$

вектор расстояний для дуги истинной информационной зависимости (a_j, a_{j+1}) после скашивания будет иметь вид

$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \times \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

Метод гиперплоскостей (Loop Wavefront) [4] — это преобразование гнезда циклов, которое меняет обход точек пространства итераций следующим образом. Пространство итераций разбивается на параллельные гиперплоскости (с общим вектором нормали). Гиперплоскости обходятся в том порядке, который указывает вектор нормали. Точки, находящиеся на одной гиперплоскости, обходятся в лексикографическом порядке.

В данной работе, для преобразования гнёзд итерационного типа, метод гиперплоскостей будет применяться поэтапно к парам соседних циклов.

Преобразование метод гиперплоскостей является комбинацией преобразований скашивание и перестановка циклов [1]. Применяется к паре соседних циклов (один из которых непосредственно вложен во второй). Условие эквивалентности метода гиперплоскостей сводится к условию эквивалентности перестановки циклов после скашивания. Если все информационные зависимости в гнезде циклов имеют векторы

расстояний и эти векторы не имеют отрицательные координаты, то перестановка циклов эквивалентна [2].

Метод гиперплоскостей выполняется таким образом, чтобы точки находящиеся на одной гиперплоскости были информационно независимы, что позволяет выполнять их параллельно.

Тайлинг (Loop Tiling, Loop Blocking) [1, 8] – это преобразование программ, которое разбивает пространство итераций исходного гнезда цикла параллельными плоскостями (гиперплоскостями) на блоки (тайлы) меньшего размера и просматривает точки пространства итераций поблочно. Прямоугольный тайлинг – это тайлинг у которого блоки являются прямоугольными параллелепипедами, грани которых перпендикулярны соответствующим координатным осям, см. рисунок 4.

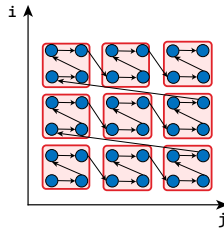


Рисунок 4. Пространство итераций двумерного гнезда циклов после тайлинга. Дуги показывают порядок выполнения точек пространства итераций. Точки, принадлежащие одному тайлу обведены.

Чтобы прямоугольный тайлинг оставался эквивалентным преобразованием должны отсутствовать информационные зависимости, для которых вектор расстояний имеет отрицательные координаты [2].

Листинг 3. Гнездо циклов полученное после применения тайлинга с размерами блоков d_1 , d_2 к двумерному гнезду циклов

```

for ( ii = 0; ii < N1/d1; ii++)
  for ( jj = 0; jj < N2/d2; jj++)
    for ( i = ii*d1; i < (ii+1)*d1; i++)
      for ( j = jj*d2; j < (jj+1)*d2; j++)
        LoopBody(i, j);

```

Скошенный тайлинг (Skewed Loop Tiling) [1] – преобразование, которое является комбинацией скашивания и прямоугольного тайлинга (см. рисунок 5).

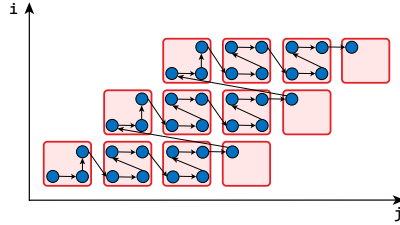


Рисунок 5. Пространство итераций двумерного гнезда циклов после скошенного тайлинга. Дуги показывают порядок выполнения точек пространства итераций. Точки, принадлежащие одному тайлу, обведены.

Листинг 4. Гнездо циклов полученное после применения скошенного тайлинга с размерами блоков d_1 , d_2 и параметром скашивания = 1 к двумерному гнезду циклов

```

for (ii=0; ii < (N-initN - 1)/d1 + 1; ii++)
  for (jj=0; jj < ((M-initM) + (N-initN) - 2)/d2 + 1; jj++)
    for (i=ii*d1; i < min((ii+1)*d1, N-initN); i++)
      for (jjj=max(i, jj*d2);
           jjj < min((jj+1)*d2, (M-initM)+i); jjj++)
        LoopBody(i, jjj - i);

```

Гнёзда циклов, для которых прямоугольный тайлинг не применим, можно преобразовывать, используя скошенный тайлинг, поскольку за счёт скашивания изменяются векторы расстояний информационных зависимостей.

Инвариант цикла – это выражение, значение которого не меняется при каждом прохождении цикла. Предварительное вычисление инвариантов до выполнения цикла может увеличить производительность.

Вынос инвариантных выражений из циклов – это преобразование, которое находит инварианты в цикле и выносит их за пределы цикла.

2. Алгоритм оптимизации гнёзд циклов итерационного типа

Опишем процесс оптимизации гнёзд циклов с применением приведённых ранее преобразований. На вход алгоритму подается программа на

языке C, содержащая тесное гнездо циклов итерационного типа.

Алгоритм оптимизации итерационных гнёзд циклов:

- (1) Вычисление параметров скашивания на основе анализа информационных зависимостей;
- (2) Применение скашивания (если необходимо) и тайлинга;
- (3) Перестановка циклов внутри тайла для повышения временной локальности данных;
- (4) Применение метода гиперплоскостей;
- (5) Применение прагм OpenMP для параллельного выполнения тайлов.

Результатом применения алгоритма является преобразованная программа на языке C.

Автоматизация описанного алгоритма реализована в Оптимизирующей распараллеливающей системе (ОРС)³.

2.1. Вычисление параметров преобразований на основе анализа информационных зависимостей


Для определения подходящего алгоритма тайлинга (прямоугольный или скошенный) проводится анализ дуг графа информационных зависимостей. Для каждой дуги вычисляется вектор расстояний (distance vector). Если хотя бы в одном векторе расстояний присутствуют отрицательные координаты, то перед применением тайлинга выполняется скашивание.

2.2. Вычисление матрицы скашивания

Пусть на графе информационных зависимостей некоторого гнезда циклов присутствуют k дуг информационной зависимости. Анализируются векторы расстояний этих дуг.

Рассмотрим дугу информационной зависимости (u, v) .

Пусть n – количество циклов в гнезде. Определим массив $\text{carriers}(u, v)$ носителей [12] дуги (u, v) . Поскольку рассматриваются тесные гнёзда циклов, все вхождения переменных находятся в самом внутреннем цикле. Это означает, что для информационных зависимостей, которые порождены этими вхождениями, все циклы гнезда могут являться их носителями. Это означает, что элементы массива $\text{carriers}(u, v)$ принимают значения от нуля до $n - 1$, нумерация циклов производится от внешнего к внутреннему.

³Оптимизирующая распараллеливающая система:  <http://www.ops.rsu.ru/>.

Пусть $\text{dist}_m = \text{dist}(u, v)_m$, где m от 0 до $n - 1$, номер отрицательной координаты вектора расстояний дуги (u, v) . В таком случае нужно выполнить скашивание циклов с номерами, которые являются элементами массива $\text{carriers}(u, v)$, относительно цикла с номером m ; параметр скашивания $f = |\text{dist}_m|$. В результате получим композицию скашиваний. Матрица этой композиции – s . Пусть S множество таких матриц скашиваний, построенных для каждой дуги информационной зависимости. Тогда сформируем результирующую матрицу скашивания skew элементы которой вычисляются по формуле: $\text{skew}_{i,j} = \max(s^1_{i,j}, s^2_{i,j}, \dots, s^k_{i,j})$, где s^1, s^2, \dots, s^k принадлежат множеству S .

ПРИМЕР 2. *Рассмотрим вычисление матрицы скашивания для гнезда циклов, представленного на рисунке 1. В данном гнезде присутствуют 5 дуг графа информационных зависимостей, которые могут повлиять на эквивалентность выполнения тайлинга: две дуги антивисимости, две дуги истинной зависимости и одна дуга выходной зависимости. Вычислим для них векторы расстояний (dist) и массивы носителей зависимости (carriers):*

Дуга информационной зависимости	Вектор расстояний информационной зависимости (dist)	Массив носителей информационной зависимости (carriers)
$(u_{i,j}, u_{i,j})$	$(1, 0, 0)$	(0)
$(u_{i,j}, u_{i-2,j-1})$	$(1, 2, 1)$	$(0, 1)$
$(u_{i-2,j-1}, u_{i,j})$	$(1, -2, -1)$	(0)
$(u_{i,j}, u_{i+1,j-1})$	$(1, -1, 1)$	(0)
$(u_{i+1,j-1}, u_{i,j})$	$(1, 1, -1)$	$(0, 1)$

Векторы расстояний дуг

$$(u_{i-2,j-1}, u_{i,j}), \quad (u_{i,j}, u_{i+1,j-1}), \quad (u_{i+1,j-1}, u_{i,j})$$

имеют отрицательные координаты. Вычислим для каждой дуги матрицу скашивания.

Вектор расстояний дуги $(u_{i-2,j-1}, u_{i,j})$ имеет две отрицательные координаты

$$\text{dist}(u_{i-2,j-1}, u_{i,j})_1 = -2, \quad \text{dist}(u_{i-2,j-1}, u_{i,j})_2 = -1.$$

Носитель информационной зависимости – цикл под номером 0. Значит нужно выполнить два скашивания: цикла под номером 0 относительно цикла под номером 1 с параметром скашивания 2 (матрица скашивания

$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$); цикла под номером 0 относительно цикла под номером 2 с параметром скашивания 1 (матрица скашивания $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$). Тогда матрица композиции скашиваний равняется:

$$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}.$$

После скашивания с такой матрицей все координаты вектора расстояний рассматриваемой дуги станут неотрицательными:

$$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 \\ -2 \\ -1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}.$$

ЗАМЕЧАНИЕ 1. Для любой матрицы скашивания, у которой элементы ниже главной диагонали не меньше, чем элементы полученной матрицы скашивания тоже приведёт к неотрицательным координатам вектора расстояний.

Вектор расстояний дуги $(u_{i,j}, u_{i+1,j-1})$ имеет одну отрицательную координату $\text{dist}(u_{i,j}, u_{i+1,j-1})_1 = -1$. Носитель информационной зависимости – цикл под номером 0. Значит нужно выполнить скашивание цикла под номером 0 относительно цикла под номером 1 с параметром скашивания 1. Матрица скашивания = $\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$. После скашивания с такой матрицей все координаты вектора расстояний рассматриваемой дуги станут неотрицательными:

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}.$$

Вектор расстояний дуги $(u_{i+1,j-1}, u_{i,j})$ имеет одну отрицательную координату $\text{dist}(u_{i+1,j-1}, u_{i,j})_2 = -1$. Носители информационной зависимости – циклы под номером 0 и 1. Значит нужно выполнить два скашивания: цикла под номером 0 относительно цикла под номером 2 с параметром скашивания 1 (матрица скашивания $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$); цикла под номером 1 относительно цикла под номером 2 с параметром скашивания 1 (матрица скашивания $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$); Тогда матрица композиции скашиваний = $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$.

После скашивания с такой матрицей все координаты вектора расстояний рассматриваемой дуги станут неотрицательными:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

Получим результирующую матрицу скашивания выбрав максимальные значения для каждой пары координат матриц

$$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} : \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}.$$

ЗАМЕЧАНИЕ 2. Скашивание должно применяться по строкам матрицы скашивания (по строкам, сверху-вниз). Так например, для матрицы скашивания $\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$:

- (1) применяется скашивание с матрицей $\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$;
- (2) применяются скашивания с матрицами $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$ и $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$ в произвольном порядке.

2.3. Применение тайлинга

Основываясь на анализе информационных зависимостей применяется прямоугольный тайлинг в комбинации со скашиванием, если требуется.

Параметрами прямоугольного тайлинга является целочисленный массив размеров тайлов (`tile_sizes`), который задаётся пользователем. Размер массива `tile_sizes` равен количеству циклов гнезда.

Параметры для скошенного тайлинга: целочисленный массив размеров тайлов (`tile_sizes`), матрица скашивания (`skew`). В начале к циклам гнезда применяется композиция скашиваний, определяемая матрицей `skew`. Затем применяется прямоугольный тайлинг с размерами тайлов (`tile_sizes`).

2.4. Перестановка циклов внутри тайла для повышения временной локальности данных

Для повышения временной локальности данных при вычислении тайла используется преобразование «перестановка циклов» (Loop Interchange). Его условия эквивалентности описаны в [2]. В данном случае, после применения тайлинга, условия эквивалентности выполняются (т.к. все векторы расстояний не будут иметь отрицательных координат).

Выбираются циклы, отвечающие за обход тайла. Определяется самый вложенный цикл (innermost loop) и переставляется с вышестоящими посредством преобразования «перестановка циклов», чтобы выбранный цикл стал внешним. До перестановки обход точек тайла проводился по целым точкам сечений тайла параллельным нижней грани. После перестановки обход производится по целым точкам сечений тайла параллельным боковой грани ближайшей к началу координат.

На рисунке (6) проиллюстрировано изменение порядка обхода итераций, которое получается после применения описанной перестановки.

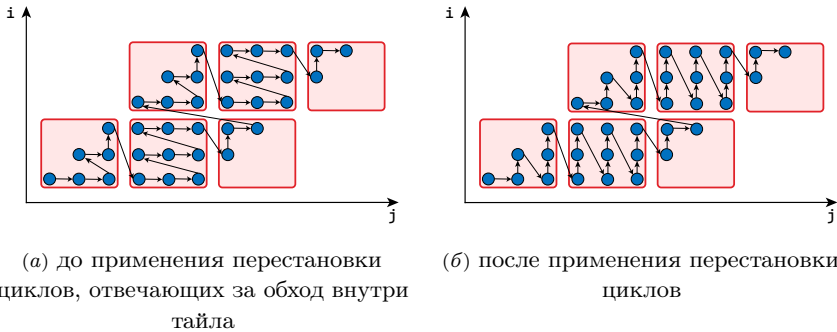


Рисунок 6. Порядок выполнения итераций двумерного гнезда циклов; точки – точки пространства итераций, дуги – порядок выполнения точек; красным выделены блоки (тайлы)

Данное преобразование может повышать временную локальность за счёт того, что вычисленные на предыдущих итерациях данные используются на следующих до того, как будут вытеснены из кэш-памяти и регистров. Ускорение, которое достигается описанной перестановкой, демонстрируется в параграфе 3.5 на примере алгоритма Гаусса-Зейделя для решения обобщённой задачи Дирихле уравнения Лапласа. Для данной задачи была выбрана матрица скашивания $skew = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$.

Для данной задачи определим откуда считываются данные при вычислении одного тайла без перестановки и с перестановкой. Обозначим M время чтения из оперативной памяти, C - из кэш-памяти и R из регистров.

Для алгоритма Гаусса-Зейделя двумерной задачи Дирихле уравнения Лапласа, на каждой итерации производится вычисление элемента массива на основе его соседних элементов

$$u_{i,j} = \frac{u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1}}{4}.$$

Таким образом выполняется 4 чтения из памяти.

Модель чтения данных для первого случая, когда обход точек тайла осуществляется по горизонтальным плоскостям (сечениям), представлена в таблицах 1, 2.

Модель чтения данных для второго случая, при котором обход точек

Таблица 1. Чтение данных первого сечения при выполнении по горизонтальным сечениям

	Чтения	Количество точек
Первая выполняемая точка тайла	$4M$	1
Край сечения – нижний	$R + 3M$	$a - 1$
Край сечения – левый	$2C + 2M$	$b - 1$
Внутренние точки сечения	$2C + R + M$	$ab - (a + b - 1)$

Таблица 2. Чтение данных последующих сечений при выполнении по горизонтальным сечениям

	Чтения	Количество точек
Первая выполняемая точка тайла	$2C + 2M$	$c - 1$
Край сечения – нижний	$R + C + 2M$	$(c - 1)(a - 1)$
Край сечения – левый	$3C + M$	$(c - 1)(b - 1)$
Внутренние точки сечения	$R + 3C$	$(c - 1)(a - 1)(b - 1)$

тайла осуществляется по плоскостям (сечениям) параллельным боковой грани тайла, представлена в таблицах 3, 4.

Таблица 3. Чтение данных первого сечения при выполнении по боковым сечениям

	Чтения	Количество точек
Первая выполняемая точка тайла	$4M$	1
Край сечения – нижний	$C + 3M$	$a - 1$
Край сечения – левый	$2R + 2M$	$b - 1$
Внутренние точки сечения	$C + 2R + M$	$ab - (a + b - 1)$

Во втором случае, при обходе точек тайла по боковым сечениям, для внутренних точек будет больше чтений данных их регистров, чем в первом случае. Это увеличивает временную локальность данных и даёт ускорение.

2.5. Применение метода гиперплоскостей и прагм OpenMP для параллельного выполнения тайлов

Основная идея алгоритма, ускоряющего вычисление гнёзд циклов итерационного типа, состоит в том, чтобы параллельно выполнять не отдельные точки пространства итераций, а блоки (тайлы) таких точек. На начальных этапах алгоритма производится анализ информационных

ТАБЛИЦА 4. Чтение данных последующих сечений при выполнении по боковым сечениям

	Чтения	Количество точек
Первая выполняемая точка тайла	$2C + 2M$	$c - 1$
Край сечения – нижний	$2C + 2M$	$(c - 1)(a - 1)$
Край сечения – левый	$2R + C + M$	$(c - 1)(b - 1)$
Внутренние точки сечения	$2R + 2C$	$(c - 1)(a - 1)(b - 1)$

зависимостей и, если требуется, выполняется скачивание. Получаем такое гнездо циклов, в котором для каждой информационной зависимости вектор расстояний не имеет отрицательных координат, что позволяет применять тайлинг.

После выполнения тайлинга, можно концептуально построить фактор-граф решетчатого графа по тайлам, т. е. вершинами такого фактор-графа являются тайлы. Дуга между вершинами фактор графа существует, если между точками тайлов, соответствующих этим вершинам, присутствует дуга информационной зависимости.

Покрываем вершины фактор-графа семейством параллельных гиперплоскостей. Удобно брать семейство гиперплоскостей с вектором нормали, имеющим все координаты единицы: $(1, 1, \dots, 1)$ (рисунок 7), поскольку

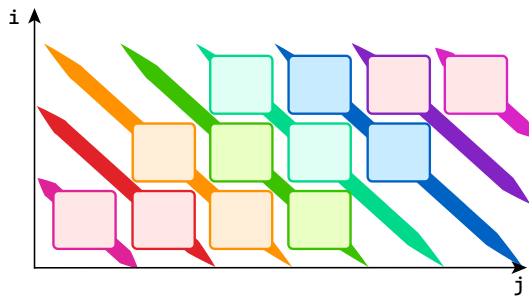


Рисунок 7. Фактор-граф по тайлам. Блоки – вершины фактор-графа. Фактор-граф покрыт семейством параллельных гиперплоскостей с вектором нормали $= (1, 1)$. Блоки, находящиеся на одной гиперплоскости, выделены одним цветом.

тогда получаются гиперплоскости более насыщенные тайлами, что выгодно для распараллеливания. Приведём обоснование того, что тайлы, лежащие в таких гиперплоскостях, попарно не связаны информационными

ЗАВИСИМОСТЯМИ.

ТЕОРЕМА 1. *После применения тайлинга тайлы, лежащие на одной гиперплоскости с вектором нормали $(1, 1, \dots, 1)$, попарно не связаны информационными зависимостями, и, следовательно, могут выполняться параллельно.*

ДОКАЗАТЕЛЬСТВО. Рассмотрим двумерный случай. Предположим, что существует пара, лежащих в одной гиперплоскости, информационно зависимых вершин фактор-графа решетчатого графа по тайлам. Тогда в соответствующих тайлах есть точки $u = (a, b), v = (c, d)$ пространства итераций, между которыми есть дуга решетчатого графа (u, v) , соответствующая некоторой дуге графа информационных зависимостей. Эта дуга решетчатого графа имеет вектор расстояния $\text{dist}(u, v) = (c - a, d - b)$. Покажем, что этот вектор будет иметь отрицательную координату.

Пусть d_1, d_2 - размеры тайлов, $(x_1, y_1), (x_2, y_2)$ - координаты этих тайлов на фактор-графе к которым принадлежат вхождения u, v соответственно, тогда выполняются неравенства:

$$\begin{aligned} x_1 d_1 &\leq a < (x_1 + 1) d_1 & y_1 d_2 &\leq b < (y_1 + 1) d_2 \\ x_2 d_1 &\leq c < (x_2 + 1) d_1 & y_2 d_2 &\leq d < (y_2 + 1) d_2 \end{aligned}$$

Поскольку $(x_1, y_1), (x_2, y_2)$ координаты тайлов лежащих на одной гиперплоскости с вектором нормали $(1, 1)$, $(x_2 - x_1) + (y_2 - y_1) = 0$. Обозначим $n = x_2 - x_1$, тогда $y_2 - y_1 = -n$,

$$\begin{aligned} x_1 d_1 &\leq a < (x_1 + 1) d_1 & y_1 d_2 &\leq b < (y_1 + 1) d_2 \\ (x_1 + n) d_1 &\leq c < ((x_1 + n) + 1) d_1 & (y_1 - n) d_2 &\leq d < ((y_1 - n) + 1) d_2 \end{aligned}$$

Из этого при $n \geq 1$ следует, что $a < c$ и $b > d$, т.е. $c - a > 0$ и $d - b < 0$. При $n < 1$ выполняется $a > c$ и $b < d$ т.е. $c - a < 0$ и $d - b > 0$. Получается, что одна из координат вектора расстояний будет отрицательной.

□

СЛЕДСТВИЕ 1. *Применение метода гиперплоскостей с вектором нормали $(1, 1, \dots, 1)$ и последующее распараллеливание является эквивалентными.*

Для того чтобы покрыть вершины фактор-графа семейством параллельных гиперплоскостей, с выбранным вектором нормали $(1, 1, \dots, 1)$, выполняется следующая последовательность действий.

- (0) Для гнезда циклов (размерности $n + 1$), после применения тайлинга (размерность гнезда $2(n + 1)$), выбираются циклы, отвечающие за обход тайлов (первые $n + 1$ циклов преобразованного гнезда).
- (1) Из выбранных циклов определяется внутренний, с порядковым номером n (нумерация циклов от 0). Переменной K присвоим номер n .
- (2) Преобразование Loop Wavefront, с входным параметром равным 1 (параметр скашивания), применимо к циклам под номерами K , $K - 1$. Порядковый номер выбранного цикла K , после применения преобразования, будет равняться $K - 1$.
- (3) Если порядковый номер выбранного цикла K не равняется нулю, то переходим к пункту (2).
- (4) Циклы с номерами от 1 до n отвечают за обход тайлов лежащих на одной гиперплоскости. К ним можно применить распараллеливание, например, за счёт добавления прагм OpenMP.

После применения описанного алгоритма может быть применено преобразование линеаризация выражений, которое дополнительно повышает производительность.

3. Эквивалентность алгоритма оптимизации итерационных гнёзд циклов

ТЕОРЕМА 2. *Алгоритм оптимизации итерационных гнёзд циклов является эквивалентным.*

ДОКАЗАТЕЛЬСТВО. Алгоритм состоит из преобразований: скашивание, тайлинг (гнездование циклов, перестановка циклов), перестановка циклов, метод гиперплоскостей (скашивание, перестановка циклов). Расширение алгоритма, дополняет представленный алгоритм преобразованиями «линеаризация» и «вынос инвариантных выражений». Доказательство эквивалентности алгоритма сводится к доказательству эквивалентности каждого преобразования последовательности.

Скашивание и гнездование не меняют порядок обращения к памяти, а потому являются эквивалентными. Если все информационные зависимости в гнезде циклов имеют векторы расстояний и эти векторы не имеют отрицательные координаты, то перестановка циклов эквивалентна [2, 9]. Для проверки эквивалентности тайлинга производится анализ информационных зависимостей. При наличии зависимостей, имеющих отрицательные координаты вектора расстояний, выполняется с соответствующими

параметрами скашивание так, чтобы все векторы расстояний имели неотрицательные координаты. Эквивалентность метода гиперплоскостей обосновывается теоремой 1. «Линеаризация» и «вынос инвариантных выражений» являются эквивалентными преобразованиями. Таким образом описанный алгоритм оптимизации итерационных гнёзд циклов и его расширение являются эквивалентными. \square

4. Численные эксперименты

Численные эксперименты были проведены на компьютере с процессором Intel i7-9700 (Coffee Lake), тактовая частота 3,00 GHz, 8 ядер, размер кэш-памяти: L1 — 256 Kb; L2 — 2 Mb; L3 — 12 Mb. В качестве компилятора использовался GCC-6.3.0-1 с опцией -O3. Ускорение вычислялось по формуле:

$$\text{Ускорение} = \frac{\text{Время выполнения исходной программы}}{\text{Время выполнения преобразованной программы}}.$$

4.1. Про выбор оптимальных размеров тайлов

Рассмотрим выбор оптимальных размеров тайлов для алгоритма Гаусса-Зейделя решения двумерной задачи Дирихле уравнения Лапласа (листинг 5). Размерность задачи $256 \times 4000 \times 4000$, тип данных double. Компилятор gcc, опция компилятора -O3. Применялся скошенный тайлинг с матрицей скашивания $\text{skew} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$ и перестановка циклов внутри тайла.

Листинг 5. Основное гнездо циклов алгоритма Гаусса-Зейделя для решения задачи Дирихле уравнения Лапласа

```
for (t=0; t<=T-1; t++)
  for (i=1; i<=N-2; i++)
    for (j=1; j<=N-2; j++)
      u[i][j] = (u[i-1][j]+u[i+1][j]+
                u[i][j-1]+u[i][j+1])/4.0;
```

При тайлинге ускорение достигается за счет того, что некоторые данные при вычислении точек тайла читаются много раз, причем все чтения, кроме первого происходят из кэш-памяти или регистров. Например, в коде листинга (листинг 5) к ячейке памяти, в которой хранится элемент массива $u_{3,7}$, программа обращается при значениях вектора счетчиков циклов (t, i, j) при вычислении не только элемента массива $u_{3,7}$, но и элементов $u_{2,7}$ $u_{4,7}$ $u_{3,6}$ $u_{3,8}$. Идея оптимизации вычисления гнёзд циклов

итерационного типа состоит в том, чтобы, не завершив вычисления одной итерации t , начинать вычисление следующей итерации ($t + 1$), пока в кэш-памяти находятся необходимые для вычислений данные.

Пусть d_1, d_2, d_3 – размеры тайлов двумерной задачи (трехмерное гнездо циклов). Пронаблюдаем влияние разных размеров тайлов на ускорение. Зафиксируем $d_2 = 50$, $d_1 = (32 \text{ либо } 64)$. Оценим влияние размера d_3 на ускорение (таблица 5).

Таблица 5. Влияние величины d_3 на ускорение программы из подраздела 4.1 при последовательном выполнении

d_3	$d_1 = 32$		$d_1 = 64$	
	Время выполнения (сек)	Ускорение	Время выполнения (сек)	Ускорение
4	3,823	2,915	3,643	3,059
8	3,359	3,318	3,257	3,421
16	3,142	3,547	3,028	3,680
20	3,088	3,609	2,990	3,727
25	3,064	3,637	2,951	3,777
40	3,007	3,706	2,880	3,869
50	2,984	3,735	2,869	3,884
80	2,959	3,766	2,815	3,959
100	2,932	3,801	2,813	3,961
125	2,920	3,816	2,825	3,944
200	2,899	3,844	2,755	4,045
250	2,895	3,849	2,754	4,046
400	2,907	3,833	2,754	4,046
500	2,887	3,860	2,731	4,080
1000	2,877	3,873	2,723	4,092
2000	2,876	3,875	2,709	4,114

При последовательном выполнении преобразованной программы наблюдается постепенное увеличение ускорения в пределах 1.1 секунды и при увеличении значения прирост уменьшается. Это связано с затратами на вычисление первого сечения тайла. Данные для первого сечения подгружаются из оперативной памяти, а для последующих – большая часть из кэш-памяти и регистров (таблицы 3, 4), чтобы компенсировать затраты на инициализацию значение должно быть как можно больше.

Таблица 6 иллюстрирует влияние размера d_3 при параллельном выполнении на 16 потоках.

Таблица 6. Влияние величины d_3 на ускорение программы из подраздела 4.1 при распараллеливании на 16 потоков

d_3	$d_1 = 32$		$d_1 = 64$	
	Время выполнения (сек)	Ускорение	Время выполнения (сек)	Ускорение
4	1,400	7,957	1,223	9,111
8	1,013	11,004	0,947	11,769
16	0,849	13,125	0,799	13,946
20	0,817	13,645	0,758	14,696
25	0,776	14,352	0,722	15,425
40	0,650	17,132	0,605	18,406
50	0,654	17,033	0,617	18,071
80	0,692	16,093	0,664	16,786
100	0,702	15,882	0,663	16,812
125	0,654	17,038	0,639	17,449
200	0,691	16,130	0,672	16,577
250	0,615	18,124	0,605	18,430
400	0,807	13,804	0,751	14,841
500	0,573	19,460	0,558	19,969
1000	0,970	11,483	0,891	12,512
2000	1,727	6,453	1,585	7,028

При распараллеливании наблюдается влияние значения d_3 на ускорение преобразованной программы. Наблюдается корреляция изменения ускорения для $d_1 = 32$ и $d_1 = 64$.

Предлагается теоретически оптимальное значение d_3 равным d_2 , которое подтверждается численными экспериментами, приведёнными в таблице 7.

Определим гипотезу для вычисления размеров d_1 и d_2 .

Для алгоритма Гаусса-Зейделя двумерной задачи Дирихле уравнения Лапласа тайл представляет собой целые точки параллелепипеда в пространстве $Z3$. Для вычислений, соответствующих внутренним точкам такого параллелепипеда все используемые данные уже использовались при вычислении этого же тайла. Эти ранее использованные данные находятся в кэш-памяти и некоторые даже в регистрах, если между повторными использованиями одного данного объем использованных других данных меньше объема кэш-памяти (не происходит вытеснения, данного из кэш-памяти).

Таким образом, объем данных между двумя использованиями одного и

ТАБЛИЦА 7. Результаты оптимизации программы из подраздела 4.1

Число потоков	Последовательно		16 потоков	
	Время выполнения	Ускорение	Время выполнения	Ускорение
$d_1 = 32, d_2 = 25, d_3 = 25$	3,605	3,11	0,752	14,90
$d_1 = 32, d_2 = 40, d_3 = 40$	3,075	3,64	0,649	17,26
$d_1 = 32, d_2 = 50, d_3 = 50$	2,982	3,76	0,651	17,21
$d_1 = 32, d_2 = 80, d_3 = 80$	2,954	3,79	0,677	16,55
$d_1 = 32, d_2 = 100, d_3 = 100$	3,088	3,63	0,687	16,31
$d_1 = 64, d_2 = 20, d_3 = 20$	3,373	3,32	0,707	15,84
$d_1 = 64, d_2 = 25, d_3 = 25$	3,200	3,50	0,688	16,28
$d_1 = 64, d_2 = 40, d_3 = 40$	2,852	3,93	0,600	18,67
$d_1 = 64, d_2 = 50, d_3 = 50$	2,884	3,88	0,603	18,57
$d_1 = 64, d_2 = 80, d_3 = 80$	2,914	3,84	0,652	17,19
$d_1 = 64, d_2 = 100, d_3 = 100$	3,038	3,69	0,662	16,92

того же данного должен быть не более объема кэш-памяти. Объем памяти между двумя использованиями одного и того же данного обозначим V . Если мы хотим минимизировать количество промахов к L1-кэш, то должно выполняться неравенство

$$(*) \quad V \leq |L1|$$

После перестановки циклов внутри тайла циклы гнезда обхода тайла расположены так, чтобы выполнение тайла проводилось по сечениям, параллельным боковой грани параллелепипеда с длинами ребер d_1 и d_2 .

В этом случае количество точек трех соседних сечений (данные которых участвуют при вычислении среднего сечения в алгоритме) равно $d_1 d_2 + 2(d_1 + d_2)$ и, в соответствии с (*), должно выполняться условие: $V \leq (d_1 d_2 + 2(d_1 + d_2)) = ((d_1 + 2)(d_2 + 2) - 4) \leq |L1|$

Границей сечения является прямоугольник со сторонами d_1 и d_2 . При фиксированной площади прямоугольника длина (количество точек) границы будет минимальна. Если стороны этого прямоугольника равны: $d_1 = d_2$. $d_1 = d_2 \leq \sqrt{|L1| + 4} - 2$. Здесь V — объединение целых точек трех сечений параллелепипеда (тайла).

Для процессора, на котором проводились вычисления, объем L1 = 32КБ на ядро процессора, тогда количество чисел двойной точности (double, 8 байт), которые могут находиться в L1 равно 4096. Следовательно теоретические оптимальные размеры d_1 и d_2 : $d_1 = d_2 \leq 62$ числа типа double.

Численные эксперименты показали при наиболее близких значениях размеров тайла $d_1 = 64$, $d_2 = 50$, $d_3 = 50$ ускорение 18,57, которое незначительно отличается от лучшего ускорения 18,67 при $d_1 = 64$, $d_2 = 40$, $d_3 = 40$ (таблица 7).

Таким образом подтверждается гипотеза о размерах тайлов d_1 , d_2 .

4.2. Сравнение с оптимизирующей распараллеливающей системой PLUTO

Проведено сравнение времени выполнения программ, полученных путём преобразования алгоритма Гаусса-Зейделя для решения обобщённой задачи Дирихле с использованием алгоритма, описанного в статье и реализованного в ОРС, и системой PLUTO.

Листинг 6. Основное гнездо циклов алгоритма Гаусса-Зейделя для решения обобщённой задачи Дирихле уравнения Лапласа

```

for (t=0; t<=T-1; t++)
  for (i=1; i<=N-2; i++)
    for (j=1; j<=N-2; j++)
      u[i][j] = (A[i][j]*u[i-1][j]+
                B[i][j]*u[i+1][j]+
                C[i][j]*u[i][j-1]+
                D[i][j]*u[i][j+1]+
                y0[i][j])/5.0;

```

Рассмотрен алгоритм Гаусса-Зейделя для численного решения обобщённой задачи Дирихле (листинг 6). Размерность задачи: 256 – количество шагов алгоритма, 2000×2000 – размер сетки. Тип данных float. Время выполнения исходного последовательного алгоритма: 7,0792 сек.

Наибольшее ускорение (в 16.32 раза) гнезда циклов, описанного в листинге, достигается алгоритмом, реализованным в ОРС, с размерами блоков $64 \times 50 \times 50$. В то время как программа преобразованная, используя PLUTO, показывает своё лучшее ускорение (в 11.34 раза) на блоках меньшей размерности $d_1 \times d_2 \times d_3 = 8 \times 8 \times 8$ (рисунок 8). Эта разница обусловлена тем, что разработанный алгоритм применяет перестановку циклов внутри тайлов, что повышает временную локальность данных внутри блока (тайла).

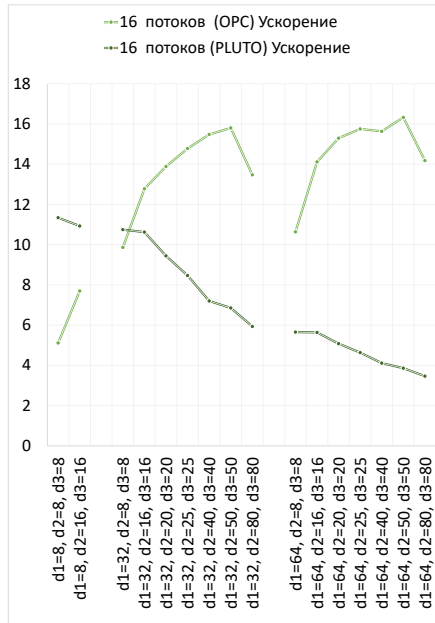


РИСУНОК 8. Сравнение ускорений при выполнении на 16 потоках, полученных при помощи PLUTO и разработанного алгоритма оптимизации (в OPC) для алгоритма Гаусса-Зейделя.

4.3. Влияние преобразований «линеаризация выражений» и «вынос инвариантных выражений» на ускорение программ

Рассмотрим влияние дополнительных преобразований «линеаризация выражений» (реализованное в OPC обобщение вычислений на этапе компиляции [10], [16]) и известное в оптимизирующей компиляции преобразование «вынос инвариантных выражений» на ускорение программ, преобразованных алгоритмом, описанным в данной статье, на примере гнезда циклов (листинг 7).

В работе [10] приведён код (более 50 строк) преобразованного необобщённого алгоритма Гаусса-Зейделя для решения задачи Дирихле уравнения Лапласа. Так же в работе [10] приведён пример применения к рассматриваемому алгоритму преобразования линеаризация выражений для упрощения выражений.

Листинг 7. Трёхмерное гнездо циклов итерационного типа, вычисления производятся по восьми соседним точкам

```

for (int k = 0; k < K; ++k )
  for (int i = 1; i < N - 1; ++i )
    for (int j = 1; j < M - 1; ++j )
      u[i][j] = (u[i-1][j]+u[i+1][j]+
                u[i][j-1]+u[i][j+1]+
                u[i-1][j-1]+u[i+1][j-1]+
                u[i+1][j-1]+u[i+1][j+1])/8.0;

```

Таким образом, за счёт применения дополнительных преобразований, после описанного алгоритма оптимизации из раздела 2, для гнёзда циклов из листинга 7, достигается дополнительное ускорение в до 1.8 раза (таблица 8).

Таблица 8. Сравнение влияния преобразований «линеаризация» и «вынос инвариантных выражений» на программу из листинга 7, предварительно преобразованную алгоритмом из раздела 2. Выполнение на 16 потоках

Размеры блоков	Время без дополнительных преобразований	Время с дополнительными преобразованиями	Ускорение
d1=32, d2=16, d3=16	3,03	1,99	1,52
d1=32, d2=20, d3=20	3,24	1,94	1,67
d1=32, d2=25, d3=25	3,39	1,92	1,77
d1=64, d2=16, d3=16	2,93	1,89	1,55
d1=64, d2=20, d3=20	3,18	1,85	1,72
d1=64, d2=25, d3=25	3,33	1,85	1,80

Заключение

В работе приводится алгоритм ускорения гнёзд циклов, основанный на комбинации оптимизирующих преобразований, и его расширение при помощи оптимизации линейных участков программы. Обосновывается эквивалентность алгоритма.

Метод изменения обхода точек тайла, который достигается перестановкой циклов внутри тайла, повышает временную локальность данных. Теоретическая модель вычисления оптимальных размеров тайлов подтверждается численными экспериментами.

Результаты численных экспериментов показывают ускорение в 16.32 раза алгоритма Гаусса-Зейделя, преобразованного при помощи приведённого алгоритма относительно исходной программы, что в 1.4 раза быстрее в сравнение с программой преобразованной известным алгоритмом оптимизации реализованном в системе PLUTO. Ускорение достигается за счёт перестановки циклов, отвечающих за обход тайлов, тем самым повышается временная локальность данных.

Численные эксперименты демонстрируют ускорение в 1.8 раза за счет применения дополнительных оптимизаций линейных участков программы, таких как линеаризация и вынос инвариантных выражений (после преобразования описанным алгоритмом оптимизации). Вынос инвариантных выражений из циклов уменьшает количество повторяющихся вычислений. Линеаризация упрощает индексные выражений и конструкции, отвечающие за вычисление границ циклов.

Список литературы

- [1] Wolf M. E., Lam M. S. *A loop transformation theory and an algorithm to maximize parallelism* // IEEE Transactions on Parallel and Distributed Systems.– 1991.– Vol. 2.– No. 4.– Pp. 452–471. [doi](#) ↑_{64, 71, 72, 73}
- [2] Wolf M., Banerjee U. *Data dependence and its application to parallel processing* // International Journal of Parallel Programming.– 1987.– Vol. 16.– No. 2.– Pp. 137–178. [doi](#) ↑_{64, 72, 77, 82}
- [3] Bondhugula U., Baskaran M., Krishnamoorthy S., Ramanujam J., Rountev A., Sadayappan P. *Automatic transformations for communication-minimized parallelization and locality optimization in the polyhedral model*, CC 2008: Compiler Construction, Lecture Notes in Computer Science.– vol. 4959, Berlin–Heidelberg: Springer.– 2008.– ISBN 978-3-540-78791-4.– Pp. 132–146. [doi](#) ↑₆₄
- [4] Lamport L. *The parallel execution of DO loops* // Commun. ACM.– 1974.– Vol. 17.– No. 2.– Pp. 83–93. [doi](#) ↑_{64, 71}
- [5] Mullanpudi R. T., Vasista V., Bondhugula U. *PolyMage: automatic optimization for image processing pipelines* // ACM SIGPLAN Notices.– 2015.– Vol. 50.– No. 4.– Pp. 429–443. [doi](#) ↑₆₄
- [6] Maydan D. E., Hennessy J. L., Lam M. S. *Efficient and exact data dependence analysis* // ACM SIGPLAN Notices.– 1991.– Vol. 26.– No. 6.– Pp. 1–14. [doi](#) ↑₆₉
- [7] Wolfe M. *Loop skewing: the wavefront method revisited* // Int. J. Parallel. Program.– 1986.– Vol. 15.– No. 4.– Pp. 279–293. [doi](#) ↑₆₉
- [8] Irigoin F., Triolet R. *Supernode partitioning* // POPL '88: Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (San Diego, California, USA, January 10–13, 1988), New York: ACM.– 1988.– ISBN 978-0-89791-252-5.– Pp. 319–329. [doi](#) ↑₇₂
- [9] Allen R., Kennedy K. *Automatic translation of FORTRAN programs to vector form* // ACM Transactions on Programming Languages and Systems.– 1987.– Vol. 9.– No. 4.– Pp. 491–542. [doi](#) ↑₈₂
- [10] Vasilenko A., Veselovskiy V., Metelitsa E., Zhivykh N., Steinberg B., Steinberg O. *Precompiler for the ACELAN-COMPOS package solvers*, PaCT 2021: Parallel Computing Technologies, Lecture Notes in Computer Science.– vol. 12942, Cham: Springer.– 2021.– ISBN 978-3-030-86359-3.– Pp. 103–116. [doi](#) ↑_{64, 88}
- [11] Штейнберг Б. Я. *О взаимосвязи между решетчатым графом программы и графом информационных связей* // Известия высших учебных заведений. Северо-Кавказский регион. Серия: Естественные науки.– 2011.– № 5(165).– С. 28–30. ✨ ↑_{66, 67}

- [12] Савельев В. А., Штейнберг Б. Я. *Распараллеливание программ*, учебник.– Ростов-на-Дону: Изд-во Южного федерального университета.– 2008.– ISBN 978-5-9275-0547-0.– 192 с. ↑_{65, 74}
- [13] Christen M., Schenk O., Burkhart H. *PATUS: a code generation and autotuning framework for parallel iterative stencil computations on modern microarchitectures // 2011 IEEE International Parallel & Distributed Processing Symposium* (Anchorage, AK, USA, 16–20 May 2011).– 2011.– Pp. 676–687. doi ↑₇₀
- [14] Bagliy A. P., Metelitsa E. A., Steinberg B. Ya. *Automatic parallelization of iterative loops nests on distributed memory computing systems*, PaCT 2023: Parallel Computing Technologies, Lecture Notes in Computer Science.– vol. **14098**, Cham: Springer.– 2023.– ISBN 978-3-031-41673-6.– Pp. 18–29. doi ↑₆₄
- [15] Воеводин В. В., Воеводин Вл. В. *Параллельные вычисления*.– СПб.: БХВ-Петербург.– 2002.– ISBN 5-94157-160-7.– 608 с. ↑₆₆
- [16] Баглий А. П., Дубров Д. В., Штейнберг Б. Я., Штейнберг Р. Б. *43–47 // Научный сервис в сети Интернет: труды XIX Всероссийской научной конференции* (18–23 сентября 2017 г., г. Новороссийск), М.: ИПМ им. М.В.Келдыша.– 2017.– ISBN 978-5-98354-037-8. URL doi ↑_{64, 88}

Поступила в редакцию	21.01.2024;
одобрена после рецензирования	13.02.2024;
принята к публикации	15.02.2023;
опубликована онлайн	26.03.2024.

Рекомендовал к публикации

к.ф.-м.н. С. А. Романенко

Информация об авторе:



Елена Анатольевна Метелица

м.н.с. Института математики механики и компьютерных наук им. Воровича, Южный федеральный университет. Научные интересы в областях компиляторных преобразований, оптимизации и распараллеливания программ



0000-0001-6253-150X

e-mail: metelica@sfnu.ru

Автор заявляет об отсутствии конфликта интересов.



Justification of methods for accelerating iterative loops nests

Elena Anatol'evna **Metelitsa**

Southern Federal University, Rostov-on-Don, Russia

 metelica@sfedu.ru

Abstract. The acceleration of iterative algorithms, found in solving problems of mathematical physics, mathematical modeling, and image processing, is considered. In the software implementation of these algorithms, there are nested loops (sections of the program that consist of nested loops). These loop nests can be accelerated by combination of optimizing transformations, including tiling, hyperplane method, and parallelization on shared memory. The equivalence of this combination of program transformations is substantiated.

A method for changing the order of tile traversal is proposed and justified. The method provides acceleration by increasing data readings from registers instead of slower memory. Considering this method, a formula for calculating optimal tile sizes is obtained.

The combination of transformations presented in this article results in an acceleration that is 1.4 times greater than the well-known optimization algorithm implemented in PLUTO. In some cases using an 8-core processor, numerical experiments show a significant increase in speed compared to the original sequential algorithms. The findings of this article can be applied to manual and automatic program optimization. (*In Russian*).

Key words and phrases: tiling, wavefront, parallelization, shared memory, iterative stencil loops


2020 *Mathematics Subject Classification:* 68W10; 68N20

Acknowledgments: The author is grateful to Dr. B.Ya. Steinberg for leadership of the work and Ar.V. Klimov for his attention and interest in the work.

For citation: Elena A. Metelitsa. *Justification of methods for accelerating iterative loops nests*. Program Systems: Theory and Applications, 2024, **15**:1(60), pp. 63–94. (*In Russ.*). https://psta.psiras.ru/read/psta2024_1_63-94.pdf

References

- [1] M. E. Wolf, M. S. Lam. “A loop transformation theory and an algorithm to maximize parallelism”, *IEEE Transactions on Parallel and Distributed Systems*, **2**:4 (1991), pp. 452–471. [doi](#)
- [2] M. Wolf, U. Banerjee. “Data dependence and its application to parallel processing”, *International Journal of Parallel Programming*, **16**:2 (1987), pp. 137–178. [doi](#)
- [3] U. Bondhugula, M. Baskaran, S. Krishnamoorthy, J. Ramanujam, A. Rountev, P. Sadayappan. “Automatic transformations for communication-minimized parallelization and locality optimization in the polyhedral model”, CC 2008: Compiler Construction, Lecture Notes in Computer Science, vol. **4959**, Springer, Berlin–Heidelberg, 2008, ISBN 978-3-540-78791-4, pp. 132–146. [doi](#)
- [4] L. Lamport. “The parallel execution of DO loops”, *Commun. ACM*, **17**:2 (1974), pp. 83–93. [doi](#)
- [5] R. T. Mullapudi, V. Vasista, U. Bondhugula. “PolyMage: automatic optimization for image processing pipelines”, *ACM SIGPLAN Notices*, **50**:4 (2015), pp. 429–443. [doi](#)
- [6] D. E. Maydan, J. L. Hennessy, M. S. Lam. “Efficient and exact data dependence analysis”, *ACM SIGPLAN Notices*, **26**:6 (1991), pp. 1–14. [doi](#)
- [7] M. Wolfe. “Loop skewing: the wavefront method revisited”, *Int. J. Parallel Program.*, **15**:4 (1986), pp. 279–293. [doi](#)
- [8] F. Irigoien, R. Triolet. “Supernode partitioning”, *POPL ’88: Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (San Diego, California, USA, January 10–13, 1988), ACM, New York, 1988, ISBN 978-0-89791-252-5, pp. 319–329. [doi](#)
- [9] R. Allen, K. Kennedy. “Automatic translation of FORTRAN programs to vector form”, *ACM Transactions on Programming Languages and Systems*, **9**:4 (1987), pp. 491–542. [doi](#)
- [10] A. Vasilenko, V. Veselovskiy, E. Metelitsa, N. Zhiviykh, B. Steinberg, O. Steinberg. “Precompiler for the ACELAN-COMPOS package solvers”, PaCT 2021: Parallel Computing Technologies, Lecture Notes in Computer Science, vol. **12942**, Springer, Cham, 2021, ISBN 978-3-030-86359-3, pp. 103–116. [doi](#)
- [11] B. Ya. Shtejnberg. “About the connection between the lattice graph and the data dependence graph”, *Izvestiya vysshix uchebnyx zavedenij. Severo-Kavkazskij region. Seriya: Estestvennye nauki*, 2011, no. 5(165), pp. 28–30 (in Russian).
- [12] V. A. Savel’ev, B. Ya. Shtejnberg. *Parallelization of Programs*, uchebnik, Izd-vo Yuzhnogo federal’nogo universiteta, Rostov-na-Donu, 2008, ISBN 978-5-9275-0547-0 (in Russian), 192 pp.
- [13] M. Christen, O. Schenk, H. Burkhart. “PATUS: a code generation and autotuning framework for parallel iterative stencil computations on modern microarchitectures”, *2011 IEEE International Parallel & Distributed Processing Symposium* (Anchorage, AK, USA, 16–20 May 2011), 2011, pp. 676–687. [doi](#)

- [14] A. P. Bagliy, E. A. Metelitsa, B. Ya. Steinberg. “Automatic parallelization of iterative loops nests on distributed memory computing systems”, PaCT 2023: Parallel Computing Technologies, Lecture Notes in Computer Science, vol. **14098**, Springer, Cham, 2023, ISBN 978-3-031-41673-6, pp. 18–29. 
- [15] V. V. Voevodin, Vl. V. Voevodin. *Parallel Computing*, BXV-Peterburg, SPb., 2002, ISBN 5-94157-160-7 (in Russian), 608 pp.
- [16] A. P. Baglij, D. V. Dubrov, B. Ya. Shtejnberg, R. B. Shtejnberg. “43–47”, *Nauchnyj servis v seti Internet: trudy XIX Vserossijskoj nauchnoj konferencii (18–23 sentyabrya 2017 g., g. Novorossijsk)*, IPM im. M.V.Keldysha, M., 2017, ISBN 978-5-98354-037-8 (in Russian). 