2024, **25** (3), 292–301. doi 10.26089/NumMet.v25r322

doi 10.26089/NumMet.v25r322

УДК 517.968;
519.642.3

Об ускоряющих преобразованиях программ для решения обобщенной задачи Дирихле

Е. А. Метелица
Южный федеральный университет,
Институт математики, механики и компьютерных наук имени И. И. Воровича,
Ростов-на-Дону, Российская Федерация
ORCID: 0000-0001-6253-150X, e-mail: metelica@sfedu.ru

Б. Я. Штейнберг
Южный федеральный университет,
Институт математики, механики и компьютерных наук имени И. И. Воровича,
Ростов-на-Дону, Российская Федерация
ORCID: 0000-0001-8146-0479, e-mail: borsteinb@mail.ru

Аннотация: В статье рассматривается цепочка преобразований программной реализации алгоритма Гаусса–Зейделя решения обобщенной двумерной задачи Дирихле уравнения Пуассона. Она дополняет прежнюю цепочку ускоряющих (в частности, распараллеливающих) преобразований этой программы. Прежняя цепочка преобразований содержала “скашивание”, “тайлинг”, “метод гиперплоскостей” и “распараллеливание”. В данной работе она дополнена преобразованиями “вынос общих подвыражений”, “вынос инвариантов цикла”, “оптимизация заголовка цикла”, “оптимизация вычисления указателей массивов”. С полученной цепочкой преобразований проведен ряд численных экспериментов на компьютере с восьмиядерным процессором. Эксперименты проводились для разных размеров тайлов. Наибольшее полученное ускорение составляет 24%. Проведен анализ полученных ускорений.

Ключевые слова: тайлинг, метод гиперплоскостей, распараллеливание, самый вложенный цикл, высокопроизводительные вычисления, обобщенная задача Дирихле.

Для цитирования: Метелица Е.А., Штейнберг Б.Я. Об ускоряющих преобразованиях программ для решения обобщенной задачи Дирихле // Вычислительные методы и программирование. 2024. **25**, № 3. 292–301. doi 10.26089/NumMet.v25r322.

2024, **25** (3), 292–301. doi 10.26089/NumMet.v25r322

On accelerating transformations of programs for solving the generalized Dirichlet problem

Elena A. Metelitsa
Southern Federal University,
Vorovich Institute of Mathematics, Mechanics, and Computer Science,
Rostov-on-Don, Russia
ORCID: 0000-0001-6253-150X, e-mail: metelica@sfedu.ru

Boris Ya. Steinberg
Southern Federal University,
Vorovich Institute of Mathematics, Mechanics, and Computer Science,
Rostov-on-Don, Russia
ORCID: 0000-0001-8146-0479, e-mail: borsteinb@mail.ru

Abstract: The chain of transformations in the program implementation of the Gauss–Seidel algorithm for solving the generalized two-dimensional Dirichlet problem of the Poisson equation is considered in this paper. It complements the previous chain of accelerating (in particular, parallelizing) transformations of this program. The previous chain of transformations contained "skewing", "tiling", "hyperplane method" and "parallelization". In this work, it is supplemented with the transformations "removal of general subexpressions", "removal of loop invariants", "optimization of the loop header", "optimization of the calculation of array pointers". A series of numerical experiments were carried out with the resulting chain of transformations on a computer with an eight-core processor. Experiments were performed for different tile sizes. The greatest obtained acceleration is 24%. An analysis of the obtained accelerations was conducted.

Keywords: tiling, wavefront, parallelization, innermost loop, high performance computing, generalized Dirichlet problem.

For citation: E. A. Metelitsa, B. Ya. Steinberg, "On accelerating transformations of programs for solving generalized Dirichlet problem," Numerical Methods and Programming. **25** (3), 292–301 (2024). doi 10.26089/NumMet.v25r322.

1. Введение. В работах [1–3] рассмотрены ускоряющие преобразования программ для гнезд циклов итерационного типа, в частности для алгоритма Гаусса–Зейделя решения задачи Дирихле уравнения Лапласа и Пуассона. Для двумерной задачи иногда достигается ускорение (относительно последовательной программы) более чем в десять раз на массовых восьмиядерных процессорах фирмы Intel.

В данной работе рассмотрен вопрос об ускорении алгоритма решения обобщенной двумерной задачи Дирихле уравнения Пуассона. Этот алгоритм представлен гнездом циклов итерационного типа на языке C (см. листинг 1).

Листинг 1. Пример решения обобщенной задачи Дирихле для уравнения Пуассона алгоритмом Гаусса–Зейделя

Listing 1. An example of solving the generalized Dirichlet problem for the Poisson equation using the Gauss–Seidel algorithm

```
1 for (int k = 0; k < K; ++k)
2     for (int i = 1; i < N - 1; ++i)
3         for (int j = 1; j < M - 1; ++j)
4             u[i][j]  =  A[i][j]*u[i-1][j] + B[i][j]*u[i+1][j] +
5                         C[i][j]*u[i][j-1] + D[i][j]*u[i][j+1] +
6                         E[i][j];
```

2024, **25** (3), 292–301. doi 10.26089/NumMet.v25r322

Представленный алгоритм сходится, если выполняется условие диагонального преобладания

$$A[i][j] + B[i][j] + C[i][j] + D[i][j] < 1,$$

где $A[i][j]$, $B[i][j]$, $C[i][j]$, $D[i][j]$ — матрицы коэффициентов. Более общие условия сходимости представлены, например, в [4].

Такие гнезда циклов могут возникать, например, в алгоритме Гаусса–Зейделя после применения метода конечных разностей к двумерной задаче Дирихле для линейного эллиптического дифференциального уравнения следующего вида

$$Q(x,y) \cdot \frac{\partial^2 u}{\partial x^2} + R(x,y) \cdot \frac{\partial^2 u}{\partial y^2} = F(x,y).$$

Ускорение представленного в листинге 1 кода происходит за счет цепочки таких преобразований гнезда циклов, как скашивание, тайлинг, метод гиперплоскостей, распараллеливание. В статье [1] в качестве дополнительного ускоряющего преобразования приводится изменение порядка обхода точек тайла за счет перестановки циклов. Там же приводится метод вычисления оптимальных размеров тайла.

Следует отметить, что в исходной программе (листинг 1) ни один цикл не допускает распараллеливание из-за информационных зависимостей. Распараллеливание возможно только после нескольких преобразований из приведенного выше списка.

Описанные выше преобразования, которые далее будем называть основными, усложняют код. Этот код может быть оптимизирован дополнительными преобразованиями.

В статье [5] показано, что оптимизирующие компиляторы (GCC, LLVM, ICC) не всегда применяют имеющиеся у них преобразования, которые могли бы ускорить программу. Проблема создания эффективного кода компилятором состоит в сложности поиска ускоряющей цепочки преобразования программ.

В [6] предлагается применять к оптимизируемой программе предварительные оптимизирующие преобразования. Если такие предварительные преобразования выполнять, получая на выходе программу на высокоуровневом языке программирования (в данном случае C), то затем можно применять компилятор даже с закрытым кодом (например, Intel C/C++ compiler). Такие предварительные преобразования можно выполнять вручную или такой распараллеливающей системой (source-to-source компилятором), как Rose-compiler или OPC [7]. Описания многих преобразований можно найти в книгах [8, 9] по распараллеливающей компиляции, в компиляторах с открытым кодом [10] или статьях по оптимизации программ, которые продолжают публиковаться.

В данной статье на примере алгоритма решения обобщенной двумерной задачи Дирихле уравнения Пуассона предлагается цепочка дополнительных ускоряющих преобразований. Действие рассматриваемых дополнительных преобразований иллюстрируется листингами. Эффективность приводимой цепочки преобразований анализируется на основе проведенных численных экспериментов.

2. Об основной цепочке преобразований. В [1, 3] рассматривается цепочка ускоряющих преобразований для гнезд циклов итерационного типа, к которым относится и код листинга 1. В эту цепочку входят “скашивание” [11], “тайлинг” [12, 13], “распараллеливание (OpenMP)”, “метод гиперплоскостей” [14]. Во многих случаях ускорение получается более, чем на порядок. В [1] к этой цепочке добавлена перестановка циклов, меняющая порядок обхода точек тайла, что ускоряет код еще приблизительно на 50% (в 1.5 раза). Эту цепочку преобразований будем называть основной.

Основная цепочка преобразования описана в работе [1]. За счет изменения информационных зависимостей скашивание позволяет подготовить программу к применению тайлинга. Тайлинг разбивает выполнение программы на блоки (тайлы), что улучшает локальность данных в кэш-памяти и подготавливает программу к распараллеливанию. Метод гиперплоскостей, примененный к циклам, отвечающим за обход тайлов, позволяет параллельно выполнять тайлы, находящиеся на одной гиперплоскости. Перестановка циклов внутри тайла улучшает локальность данных.

В [15] отмечено, что производительность вычислительных операций растет со скоростью 30% в год, а производительность памяти — 9%. Разумеется, эти цифры приблизительны и изменения происходят скачкообразно, но, тем не менее, они подтверждались несколько десятилетий. Массовые многоядерные процессоры Intel соответствуют этой тенденции. В отмеченной цепочке ускоряющих преобразований тайлинг направлен на оптимизацию использования памяти, скашивание — вспомогательное преобразование, необходимое для выполнения тайлинга, распараллеливание (OpenMP) направлено на ускорение выполнения вычислительных операций, метод гиперплоскостей — вспомогательное преобразования для ускорения.

2024, **25** (3), 292–301. doi 10.26089/NumMet.v25r322

Изменение порядка обхода точек тайла (перестановка циклов) приводит к чтению данных из более быстрых модулей компьютера (регистры вместо кэш-памяти или кэш-память вместо оперативной).

3. Дополнительные преобразования ускоренного алгоритма решения обобщенной задачи Дирихле. В ОРС (оптимизирующей распараллеливающей системе) реализована автоматизированная цепочка описанных в предыдущем разделе основных преобразований: скашивание, тайлинг, метод гиперплоскостей, распараллеливание, перестановка циклов внутри тайла. После такой цепочки преобразований количество строк исходного кода увеличивается на порядок. В работе [6] приведен код (более 50 строк) преобразованного необобщенного алгоритма Гаусса–Зейделя для решения задачи Дирихле уравнения Лапласа. В частности, гнездо из трех циклов преобразуется в гнездо из шести циклов. В первую очередь оптимизировать следует самые глубоко вложенные циклы (innermost loops), поскольку их операции вызываются чаще других.

Опишем реализованную в данной работе цепочку дополнительных преобразований для самого глубоко вложенного цикла. Для второго по вложенности цикла далее будут намечены оптимизирующие преобразования, направленные на упрощение условных операторов.

Приведем (листинг 2) полученный после основной цепочки преобразований код, упрощенный для удобства чтения, убрав созданные в ОРС:

а) фигурные скобки, заключающие один оператор;

б) прибавления и вычитания 0;

в) умножения и деления на 1.

В этом листинге остаются автоматически сгенерированные при выполнении преобразований ОРС сложные имена переменных, которые могут включать в себя, кроме имен переменных исходной программы, еще числа и подчерки.

Листинг 2. Фрагмент программы алгоритма Гаусса–Зейделя решения обобщенной задачи Дирихле, преобразованной при помощи основных преобразований

Listing 2. Fragment of Gauss–Seidel algorithm program for solving the generalized Dirichlet problem, transformed using basic transformations

```
for (k = __uni40k; k < __uni41k; k = k + 1) {
  int __uni38__uni30i, __uni39__uni30i;
  if (k > __uni32__uni30i * d2)
    __uni34__uni30i = k;
  else
    __uni34__uni30i = __uni32__uni30i * d2;
  if (((N - 1) - 1) + k < (__uni32__uni30i + 1) * d2)
    __uni35__uni30i = ((N - 1) - 1) + k;
  else
    __uni35__uni30i = (__uni32__uni30i + 1) * d2;
  if ((__uni29__uni28j + 1) < __uni35__uni30i)
    __uni39__uni30i = (__uni29__uni28j + 1);
  else
    __uni39__uni30i = __uni35__uni30i;
  if ((((__uni29__uni28j - ((M - 1) - 1)) ) + 1) > __uni34__uni30i)
    __uni38__uni30i = (((__uni29__uni28j - ((M - 1) - 1)) ) + 1);
  else
    __uni38__uni30i = __uni34__uni30i;
  for (uni30i = __uni38__uni30i; uni30i < __uni39__uni30i; uni30i = uni30i + 1) {
    i = uni30i - k;
    __uni28j = __uni29__uni28j - k;
    j = __uni28j - i;
    u[1+i][1+j] = A[i+1][j+1]*u[(1+i)-1][1+j]+B[i+1][j+1]*u[(1+i)+1][1+j]+
            C[i+1][j+1]*u[1+i][(1+j)-1]+D[i+1][j+1]*u[1+i][(1+j)+1]+E[i+1][j+1];
  }
}
```

2024, **25** (3), 292–301. doi 10.26089/NumMet.v25r322

Заменим в листинге 3 выражения `(1+i)` и `(1+j)` новыми переменными `ii` и `jj` соответственно, сократив количество сложений. При этом `i` заменяется выражением `(ii-1)` (листинг 4).

Листинг 3. Самый вложенный цикл листинга 2
Listing 3. The most nested loop of the listing 2

```
for (uni30i = __uni38__uni30i; uni30i < __uni39__uni30i; uni30i = uni30i + 1) {
  i = uni30i - k;
  __uni28j = __uni29__uni28j - k;
  j = __uni28j - i;
  u[1+i][1+j] = A[i+1][j+1]*u[(1+i)-1][1+j]+B[i+1][j+1]*u[(1+i)+1][1+j]+
            C[i+1][j+1]*u[1+i][(1+j)-1]+D[i+1][j+1]*u[1+i][(1+j)+1]+E[i+1][j+1];
}
```

Листинг 4. Результат замены выражений `(1+i)` и `(1+j)` новыми переменными `ii` и `jj` соответственно
Listing 4. The result of replacing of expressions `(1+i)` and `(1+j)` with new variables `ii` and `jj`, respectively

```
for (uni30i = __uni38__uni30i; uni30i < __uni39__uni30i; uni30i = uni30i + 1) {
    ii = uni30i - k + 1;
    __uni28j = __uni29__uni28j - k;
    jj = __uni28j - (ii - 1) + 1;
    u[ii][jj] = A[ii][jj]*u[ii-1][jj]+B[ii][jj]*u[ii+1][jj]+
                C[ii][jj]*u[ii][jj-1]+D[ii][jj]*u[ii][jj+1]+E[ii][jj];
}
```

Заменим в заголовке цикла счетчик `uni30i` новым счетчиком `ii`, уменьшив тело цикла на один оператор присваивания (листинг 5).

Листинг 5. Результат подстановки переменной `ii` на место счетчика циклов `uni30i`
Listing 5. Result of substituting the variable `ii` in place of the `uni30i` loop counter

```
for (ii = __uni38__uni30i - k + 1; ii < __uni39__uni30i - k + 1; ii = ii + 1) {
    __uni28j = __uni29__uni28j - k;
    jj = __uni28j - ii +1;
    u[ii][jj] = A[ii][jj]*u[ii-1][jj]+B[ii][jj]*u[ii+1][jj]+
                C[ii][jj]*u[ii][jj-1]+D[ii][jj]*u[ii][jj+1]+E[ii][jj];
}

```

Оптимизируем заголовок цикла и вынесем вычисление переменной `__uni28j` за пределы цикла, поскольку эта переменная не зависит от счетчика цикла (листинг 6).

Уменьшим количество операций при вычислении адресов двумерных массивов, которые являются коэффициентами при вычисляемой переменной `u`. Выполним оптимизацию вычисления указателей этих массивов. Так как в вычислениях (листинг 6) участвует 6 массивов одинаковой размерности, можно заранее вычислить положение их указателей. В случае если элементы массива расположены подряд, запись `u[ii][jj]` эквивалентна `*(*u + ii * M + jj)`, где `M` — вторая размерность двумерного массива. Вычисление позиции `[ii][jj]` можно вынести в переменную. Листинг 7 демонстрирует результат такой оптимизации. Для того чтобы данное преобразование было корректным, требуется разместить данные массивов в памяти подряд. Размещение приводится в листинге 8.

4. Результаты численных экспериментов. Численные эксперименты были проведены на компьютере с 8-ядерным процессором Intel i7-9700 (Coffee Lake), тактовая частота которого 3.00 ГГц, а размер кэш-памяти имеет следующие показатели: L1 — 256 КБ; L2 — 2 МБ; L3 — 12 МБ. В качестве компилятора

2024, **25** (3), 292–301. doi 10.26089/NumMet.v25r322

Листинг 6. Результат выноса вычисления переменной `__uni28j` за пределы внутреннего цикла и оптимизации заголовка цикла

Listing 6. The result of moving the calculation of variables `__uni28j` out from the inner loop and optimization of the loop header

```
__uni39__uni30i_k1 = __uni39__uni30i - k + 1;
__uni28j = __uni29__uni28j - k;
for (ii = __uni38__uni30i - k + 1; ii < __uni39__uni30i_k1; ii = ii + 1) {
    jj = __uni28j - ii +1;
    u[ii][jj] = A[ii][jj]*u[ii-1][jj]+B[ii][jj]*u[ii+1][jj]+
                C[ii][jj]*u[ii][jj-1]+D[ii][jj]*u[ii][jj+1]+E[ii][jj];
}
```

Листинг 7. Результат оптимизации указателей массивов внутреннего цикла

Listing 7. The result of optimizing the array pointers of the innermost loop

```
__uni39__uni30i_k1 = __uni39__uni30i - k + 1;
__uni28j = __uni29__uni28j - k;
for (ii = __uni38__uni30i - k + 1; ii < __uni39__uni30i_k1; ii = ii + 1) {
    jj = __uni28j - ii +1;
    int ind = ii * M + jj;
    *(*u+ind) = *(*A+ind)*(*(*u+ind-N))+*(*B+ind)*(*(*u+ind - 1))+
                *(*C+ind)*(*(*u+ind+N))+*(*D+ind)*(*(*u+ind+1))+*(*E+ind);
}
```

Листинг 8. Размещение данных в памяти для оптимизации вычисления указателей массивов на примере массива `u`

Listing 8. Placing data in memory to optimize the calculation of array pointers using the example of the array `u`

```
for (int i = 0; i < N; ++i ) {
  for (int j = 0; j < M; ++j ) {
      u[i][j] = (double)rand()/(double)(RAND_MAX);
  }
}
double* data_arr = (double*)malloc(N * M * sizeof(double));
for (size_t i = 0; i < N; i++) {
  for (size_t j = 0; j < M; j++) {
    data_arr[i * M + j] = u[i][j];
  }
}
double** U = (double**)malloc(N * sizeof(double*));
for (size_t i = 0; i < N; i++) {
  U[i] = &data_arr[i * M];
}
```

использовался GCC v. 6.3.0-1 с опцией `-O3`. Ускорение вычислялось по формуле:

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

Замеры времени выполнения программы выполнялись с помощью функции `clock()` несколько раз, после чего вычислялось среднее арифметическое полученных значений. Цель численных экспериментов — изучение влияния описанных выше дополнительных преобразований на основные ускоряющие преобразования алгоритма Гаусса–Зейделя для решения обобщенной задачи Дирихле. В табл. 1 приводятся результаты численных экспериментов для значений `k=256`, `N=M=4000` и демонстрируется ускорение для

2024, **25** (3), 292–301. doi 10.26089/NumMet.v25r322

Таблица 1. Влияние дополнительных преобразований на ускорение при разных размерах тайлов

Table 1. Impact of additional transformations on speedup for different tile sizes

	Основные преобразования General transformations		Дополнительные преобразования Additional transformations		
Размеры блоков Tile sizes	Время, с Time, s	Ускорение Speed up	Время, с Time, s	Ускорение Speed up	Ускорение за счет доп. оптимизаций Speed up by additional transformations
d1=16, d2=16, d3=16	4.779	4.284	5.051	4.054	0.946
d1=16, d2=20, d3=20	4.347	4.711	4.229	4.841	1.028
d1=16, d2=25, d3=25	4.116	4.975	3.787	5.407	1.087
d1=16, d2=40, d3=40	4.576	4.474	3.723	5.499	1.229
d1=16, d2=50, d3=50	4.357	4.700	3.513	5.829	1.240
d1=16, d2=80, d3=80	3.756	5.451	3.094	6.618	1.214
d1=16, d2=100, d3=100	3.457	5.923	3.034	6.749	1.139
d1=16, d2=125, d3=125	3.602	5.684	3.721	5.502	0.968
d1=16, d2=200, d3=200	8.718	2.349	10.272	1.993	0.849
d1=32, d2=16, d3=16	4.181	4.897	4.411	4.642	0.948
d1=32, d2=20, d3=20	3.688	5.552	3.807	5.379	0.969
d1=32, d2=25, d3=25	3.350	6.113	3.274	6.254	1.023
d1=32, d2=40, d3=40	3.152	6.496	2.922	7.008	1.079
d1=32, d2=50, d3=50	3.046	6.722	2.794	7.328	1.090
d1=32, d2=80, d3=80	2.754	7.434	2.548	8.036	1.081
d1=32, d2=100, d3=100	2.798	7.317	2.614	7.833	1.071
d1=32, d2=125, d3=125	3.043	6.729	3.272	6.257	0.930
d1=32, d2=200, d3=200	8.647	2.368	10.406	1.968	0.831
d1=64, d2=16, d3=16	3.852	5.316	4.004	5.114	0.962
d1=64, d2=20, d3=20	3.210	6.379	3.380	6.058	0.950
d1=64, d2=25, d3=25	2.915	7.025	2.952	6.935	0.987
d1=64, d2=40, d3=40	2.690	7.611	2.525	8.110	1.066
d1=64, d2=50, d3=50	2.593	**7.896**	2.431	**8.424**	1.067
d1=64, d2=80, d3=80	2.691	7.608	2.482	8.251	1.085
d1=64, d2=100, d3=100	2.624	7.804	2.515	8.142	1.043
d1=64, d2=125, d3=125	2.970	6.894	3.233	6.334	0.919
d1=64, d2=200, d3=200	8.652	2.367	9.932	2.061	0.871

оптимальных размеров блоков (`d1=64`, `d2=50`, `d3=50`), при которых достигается наименьшее время выполнения расчетов после применения базовых преобразований, а также дополнительные преобразования дают ускорение на 6.7% больше относительно алгоритма, преобразованного базовыми преобразованиями, выполненными на 16 потоках. Размерность задачи — 4000×4000, количество итераций алгоритма равно 256, тип данных double. Заполнение начальными данными описано в листинге 9. Представленный алгоритм ускорения программы листинга 1 цепочкой преобразований не зависит от того, является ли соответствующая коду матрица симметричной или несимметричной.

Проводились эксперименты для сеток, пространственные размерности которых являются степенью двойки, например вместо 2000 брали 2048 или вместо 4000 — 4096. При равных степеням двойки размерностях лучших ускорений на рассматриваемом процессоре не получено.

2024, **25** (3), 292–301. doi 10.26089/NumMet.v25r322

Листинг 9. Инициализация массивов случайными начальными данными
Listing 9. Initializing arrays with initial random data

```
for (int i = 0; i < N; ++i ) {
    for (int j = 0; j < M; ++j ) {
        double temp = abs((double)rand() / (double)(RAND_MAX));
        a[i][j] = temp/2.0;
        b[i][j] = (1-temp)/2.0;
        c[i][j] = temp/2.0;
        d[i][j] = (1-temp)/2.0;
        e[i][j] = (double)rand()/(double)(RAND_MAX);
        u[i][j] = (double)rand()/(double)(RAND_MAX);
    }
}
```

Условие завершения ускоренного алгоритма, в частности сравнение невязки с заранее заданной точностью, должно вычисляться на итерациях кратных 64. Это связано с тем, что тайлы, на которые разбивается пространство итераций исходного гнезда циклов (листинг 1) в проводимых численных экспериментах, имеют количество итераций `d1` равное 16, 32 или 64. Для других размеров тайлов проверки условия завершения должны происходить через величину, кратную соответствующей размерности тайла. Количество итераций ускоренного алгоритма, необходимое для достижения заданной точности, может быть больше количества итераций исходного гнезда циклов на величину, равную размерности тайла по итерациям (величина `d1`) — это вытекает из эквивалентности каждого преобразования рассматриваемой цепочки преобразований. Эквивалентность представленных преобразований обеспечивает не только одинаковость погрешностей округлений у исходной и преобразованной программы, но и возможность реализации цепочки этих преобразований для автоматизированного выполнения компилятором.

5. Дальнейшие возможные ускоряющие преобразования. Будем рассматривать преобразование, устраняющее условный оператор внутри цикла. В условии фигурирует счетчик цикла, что позволяет разбить исходный цикл на два: с телом ветки `true` и ветки `else`. Это преобразование имеет на входе код вида, представленного в листинге 10, и заменяет этот код следующим фрагментом из листинга 11.

Листинг 10. Пример цикла с условным оператором, зависящим от счетчика цикла
Listing 10. Example of a loop with a conditional statement depending on the loop counter

```
for (k = __uni40k; k < __uni41k; k = k + 1) {
    if (k < L) {
        x = a;
    } else {
        x = b;
    }
    Block(x);
}
```

Листинг 11. Преобразованный цикл из листинга 7
Listing 11. Transformed loop from listing 7

```
for (k = __uni40k; k < L; k = k + 1)
    Block(a);
for (k = L; k < __uni41k; k = k + 1)
    Block(b);
```

6. Заключение. Полученные результаты численных экспериментов показывают, что представленные в работе дополнительные преобразования могут ускорить программу. Лучшее ускорение при оптимальных размерах тайлов достигает значения 8.4 раза, что больше ускорения основной цепочки преоб-

2024, **25** (3), 292–301. doi 10.26089/NumMet.v25r322

разований на 6.7%. Можно также заметить, что для тайлов, размеры которых не оптимальны, ускорение дополнительной цепочки преобразований может достигать 24%, хотя, в некоторых случаях, может вызывать замедление. Проведенные численные эксперименты показали, что при использовании дополнительных преобразований оптимальные размеры тайла близки к расчетным, которые представлены в [1].

Автоматический поиск лучшей цепочки ускоряющих преобразований является основной пока не решенной проблемой развития оптимизирующих компиляторов [5]. Поэтому такая цепочка должна собираться либо указаниями компилятору (прагмами), либо код следует преобразовывать вручную.

Список литературы

1. *Метелица Е.А.* Обоснование методов ускорения гнезд циклов итерационного типа // Программные системы: теория и приложения. 2024. **15**, № 1. 63–94. doi 10.25209/2079-3316-2024-15-1-63-94.
2. *Bagliy A.P., Metelitsa E.A., Steinberg B.Ya.* Automatic parallelization of iterative loops nests on distributed memory computing systems // Lecture Notes in Computer Science. Vol. 14098. Cham: Springer, 2023. 18–29. doi 10.1007/978-3-031-41673-6_2.
3. *Bondhugula U., Baskaran M., Krishnamoorthy S., et al.* Automatic transformations for communication-minimized parallelization and locality optimization in the polyhedral model // Lecture Notes in Computer Science. Vol. 4959. Berlin: Springer, 2008. 132–146. doi 10.1007/978-3-540-78791-4_9.
4. *Котина Е.Д.* О сходимости блочных итерационных методов // Известия Иркутского гос. университета. Серия: Математика. 2012. **5**, № 3. 41–55.
5. *Gong Z., Chen Z., Szaday J., et al.* An empirical study of the effect of source-level loop transformations on compiler stability // Proc. ACM on Programming Languages. 2018. **2**, N OOPSLA. Article No. 126. doi 10.1145/3276496.
6. *Vasilenko A., Veselovskiy V., Metelitsa E., et al.* Precompiler for the ACELAN-COMPOS package solvers // Lecture Notes in Computer Science. Vol. 12942. Cham: Springer, 2021. 103–116. doi 10.1007/978-3-030-86359-3_8.
7. Optimizing parallelizing system. http://www.ops.rsu.ru/. Cited August 9, 2024.
8. *Allen R., Kennedy K.* Optimizing compilers for modern architectures. San Francisco: Morgan Kaufmann, 2002.
9. *Muchnick S.S.* Advanced compiler design implementation. San Francisco: Morgan Kaufmann, 1997.
10. LLVM's analysis and transform passes. https://opensource.apple.com/source/lldb/lldb-76/llvm/docs/Passes.html. Cited August 9, 2024.
11. *Wolfe M.* Loop skewing: the wavefront method revisited // Int. J. Parallel Prog. 1986. **15** (4). 279–293. doi 10.1007/BF01407876.
12. *Irigoin F., Triolet R.* Supernode partitioning // Proc. 15th ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages. San Diego, USA, January 10–13, 1988. https://dl.acm.org/doi/pdf/10.1145/73560.73588. Cited August 10, 2024. doi 10.1145/73560.73588.
13. *Wolf M.E., Lam M.S.* A loop transformation theory and an algorithm to maximize parallelism // IEEE Trans. Parallel Distrib. Syst. 1991. **2** (4). 452–471. doi 10.1109/71.97902.
14. *Lamport L.* The parallel execution of DO loops // Commun. ACM. 1974. **17**, N 2. 83–93. doi 10.1145/360827.360844.
15. *Graham S.L., Snir M., Patterson C.A. (Eds).* Getting up to speed: the future of supercomputing. Washington, D.C.: National Academies Press, 2005. doi 10.17226/11148.

Поступила в редакцию
25 июля 2024 г.

Принята к публикации
7 августа 2024 г.

Информация об авторах

Елена Анатольевна Метелица — мл. науч. сотр.; Южный федеральный университет, Институт математики, механики и компьютерных наук имени И. И. Воровича, ул. Мильчакова, 8-А, 344090, Ростов-на-Дону, Российская Федерация.

Борис Яковлевич Штейнберг — д.т.н., ст. науч. сотр., заведующий кафедрой алгебры и дискретной математики; Южный федеральный университет, Институт математики, механики и компьютерных наук имени И. И. Воровича, ул. Мильчакова, 8-А, 344090, Ростов-на-Дону, Российская Федерация.

2024, **25** (3), 292–301. doi 10.26089/NumMet.v25r322

References

1. E. A. Metelitsa, "Justification of Methods for Accelerating Iterative Loops Nests," Program Systems: Theory and Applications, **15** (1), 63–94 (2024). doi 10.25209/2079-3316-2024-15-1-63-94.
2. A. P. Bagliy, E. A. Metelitsa, and B. Ya. Steinberg, "Automatic Parallelization of Iterative Loops Nests on Distributed Memory Computing Systems," in *Lecture Notes in Computer Science* (Springer, Cham, 2023), Vol. 14098, pp. 18–29. doi 10.1007/978-3-031-41673-6_2.
3. U. Bondhugula, M. Baskaran, S. Krishnamoorthy, et al., "Automatic Transformations for Communication-Minimized Parallelization and Locality Optimization in the Polyhedral Model," in *Lecture Notes in Computer Science* (Springer, Berlin, 2008), Vol. 4959, pp. 132–146. doi 10.1007/978-3-540-78791-4_9.
4. E. D. Kotina, "On Convergence of Block Iterative Methods," Izv. Irkutsk Gos. Univ. Ser. Matem. **5** (3), 41–55 (2012).
5. Z. Gong, Z. Chen, J. Szaday, et al., "An Empirical Study of the Effect of Source-Level Loop Transformations on Compiler Stability," Proc. ACM Program. Lang. **2** (OOPSLA), Article No. 126 (2018). doi 10.1145/3276496.
6. A. Vasilenko, V. Veselovskiy, E. Metelitsa, et al., "Precompiler for the ACELAN-COMPOS Package Solvers," in *Lecture Notes in Computer Science* (Springer, Cham, 2021), Vol. 12942, pp. 103–116. doi 10.1007/978-3-030-86359-3_8.
7. Optimizing Parallelizing System. http://www.ops.rsu.ru/. Cited August 9, 2024.
8. R. Allen and K. Kennedy, *Optimizing Compilers for Modern Architectures* (Morgan Kaufmann, San Francisco, 2002).
9. S. S. Muchnick, *Advanced Compiler Design Implementation* (Morgan Kaufmann, San Francisco, 1997).
10. LLVM's Analysis and Transform Passes. https://opensource.apple.com/source/lldb/lldb-76/llvm/docs/Passes.html. Cited August 9, 2024.
11. M. Wolfe, "Loop Skewing: The Wavefront Method Revisited," Int. J. Parallel Prog. **15** (4), 279–293 (1986). doi 10.1007/BF01407876.
12. F. Irigoin and R. Triolet, "Supernode Partitioning," in *Proc. 15th ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages, San Diego, USA, January 10–13, 1988* https://dl.acm.org/doi/pdf/10.1145/73560.73588. Cited August 10, 2024. doi 10.1145/73560.73588.
13. M. E. Wolf and M. S. Lam, "A Loop Transformation Theory and an Algorithm to Maximize Parallelism," IEEE Trans. Parallel Distrib. Syst. **2** (4), 452–471 (1991). doi 10.1109/71.97902.
14. L. Lamport, "The Parallel Execution of DO Loops," Commun. ACM. **17** (2), 83–93 (1974). doi 10.1145/360827.360844.
15. S. L. Graham, M. Snir, and C. A. Patterson (Eds.). *Getting up to Speed: The Future of Supercomputing* (National Academies Press, Washington, D.C., 2005). doi 10.17226/11148.

Received
Jule 25, 2024

Accepted for publication
August 7, 2024

Information about the authors

Elena A. Metelitsa — junior researcher; Southern Federal University, Vorovich Institute of Mathematics, Mechanics, and Computer Science, ulitsa Milchakova, 8-A, 344090, Rostov-on-Don, Russia.

Boris Ya. Steinberg — Dr. Sci., Senior Researcher, Head of the Department of Algebra and Discrete Mathematics; Southern Federal University, Vorovich Institute of Mathematics, Mechanics, and Computer Science, ulitsa Milchakova, 8-A, 344090, Rostov-on-Don, Russia.