УДК 519.612 DOI: 10.14529/mmp230105

ОБ АВТОМАТИЗАЦИИ ПРИМЕНЕНИЯ РАЗМЕЩЕНИЯ ДАННЫХ С ПЕРЕКРЫТИЯМИ В РАСПРЕДЕЛЕННОЙ ПАМЯТИ

Л.Р. Гервич[1]***, Б.Я. Штейнберг***[2]
[1]BroutonLab, г. Ростов-на-Дону, Российская Федерация
[2]Южный федеральный университет, г. Ростов-на-Дону, Российская Федерация

В статье рассматриваются блочно-аффинные размещения данных с перекрытиями для оптимизации параллельных вычислений в вычислительной системе с распределенной памятью. Примерами целевых вычислительных систем являются высокопроизводительные кластеры и перспективные системы на кристалле с большим количеством вычислительных ядер. Предлагается описывать размещение массива с перекрытиями как новый массив немного большей длины, у которого дополнительные элементы имеют значения некоторых элементов исходного массива. Рассматривается возможность разработки автоматического преобразования (компилятором) обычного размещения массива в распределенной памяти в новый массив, содержащий перекрытия. Предлагаемый метод иллюстрируется на известном численном алгоритме решения задачи теплопроводности.

Ключевые слова: автоматизация распараллеливания; распределенная память; преобразования программ; размещение данных; пересылки данных.

Введение

В данной работе предлагается метод автоматического оптимизирующего преобразования программ для распараллеливающих компиляторов на вычислительные системы с распределенной памятью.

В работе [1] отмечается: «Компиляция для параллельной архитектуры с распределенной памятью считается очень сложной задачей ... никакого практического и эффективного решения в настоящее время не существует ...». Промышленные распараллеливающие компиляторы(GCC, ICC, MS-Compiler, LLVM) также распараллеливают программы только для вычислительных систем с общей памятью. Преобразование последовательных программ на вычислительные системы с распределенной памятью требует разработки дополнительных функций, которых нет в системах с общей памятью: размещение данных в распределенной памяти, переразмещения данных в распределенной памяти (генерация межпроцессорных пересылок).

Для ВС (вычислительная система) с распределенной памятью самой длительной операцией является межпроцессорная пересылка данных. Поэтому в случае распределенной памяти основной целью оптимизации становится минимизация межпроцессорных пересылок, которые могут быть реализованы с помощью MPI, SHMEM или других инструментов. Для высокопроизводительных кластеров такие пересылки могут погасить ускорение от распараллеливания и даже вызвать замедление. В последнее время появляются многоядерные процессоры, иногда называемые «суперкомпьютер на кристалле», с десятками, сотнями и тысячами ядер [2, 3]. Пересылка данных между процессорными ядрами на одной микросхеме требует значительно меньше времени, чем на коммуникационной сети (Ethernet, Infiniband, PCI-express...). Это означает расширение множества эффективно распараллеливаемых программ и делает целесообразным и даже необходимым разработку распараллеливающих компиляторов. Новые модели параллельных вычислений на микросхемы нового типа предлагаются в [4].

В [5, 6] описаны блочно-аффинные размещения данных в распределенной памяти. Следует отметить работы группы DVM-System [7, 8] по генерации параллельного кода на ВС с распределенной памятью. В работе [9] рассмотрена задача распараллеливания программного цикла на ВС с распределенной памятью и с минимизацией количества межпроцессорных пересылок. Время, необходимое на пересылки данных нелинейно зависит от объема пересылаемых данных. Значительное время отнимает инициализация пересылки. Метод размещения данных с перекрытиями [10, 11] существенно ускоряет параллельные итерационные алгоритмы благодаря уменьшению количества пересылок при укрупнении множеств пересылаемых данных.

В [12] приводятся результаты экспериментов, показывающие, что современные оптимизирующие компиляторы плохо оптимизируют код, имеют большой неиспользованный потенциал оптимизирующих преобразований. Можно полагать, что для микросхем с сотнями и тысячами вычислительных ядер [2], этот потенциал оптимизаций больше, чем для процессоров, на которых проводились эксперименты.

Данная работа направлена на создание компилятора, который анализирует высокоуровневый текст и преобразует его к виду, допускающему распараллеливание на ВС с распределенной памятью. Предлагаемое в статье преобразование иллюстрируется на численном алгоритме решения одномерной задачи теплопроводности и может использоваться для других итерационных численных методов, например, таких как в [10].

1. Гнездование цикла

Далее понадобится используемое в компиляторах преобразование программ «гнездование цикла» (см. [13]), которое заменяет программный цикл 1.

Листинг 1. Исходный цикл

```
For (j = 1; j < = N; j = j+1) {
    LoopBody(j)
}
```

гнездом из двух циклов 2 и еще одним циклом 3 в случае, если число N не делится нацело на p:

Листинг 2. Гнездо циклов

```
For (j1 = 1; j1 < = p; j1 = j1+1) {
    For (j2 = 1; j2 < = N/p; j2 = j2+1) {
        j = (p-1)*j1+j2;
        LoopBody(j)
    }
}
```

Листинг 3. Остаточный цикл

```
For (j = p*(N//p)+1; j < = N; j = j+1) {
    LoopBody(j)
}
```

Здесь «//» – целочисленное деление, т. е. вещественное деление с отбрасыванием дробной части результата, LoopBody(j) – символическая запись тела цикла. Параметр преобразования p удобно выбирать равным количеству ПЭ (процессорных элементов)

– в этом случае при распараллеливании каждая итерация внешнего цикла выполняется в своем ПЭ. Будем полагать, что ПЭ занумерованы от 0 до $(p-1)$. Цикл 3 имеет мало итераций (меньше p) и не сильно влияет на время выполнения 1. Этот цикл также может быть выполнен последовательно либо параллельно при использовании не всех, а только $(N - p * (N//p))$ ПЭ. На ВС с распределенной памятью описанные параллельные вычисления возможны, если для каждого $j1$ все необходимые данные находятся в ПЭ с номером $j1$. Чтобы выполнилось это условие, необходимо подходящее размещение данных в распределенной памяти.

2. Блочно-аффинные размещения массивов

Основная особенность параллельного выполнения цикла на ВС с распределенной памятью состоит в том, что для каждой операции ее аргументы должны быть в одном модуле распределенной памяти.

Будем полагать, что все процессорные элементы (ПЭ) занумерованы, начиная с нуля. Размещение массива в памяти – это функция, которая для каждого элемента массива возвращает номер ПЭ, в котором этот элемент находится [5, 6]. Обычно при описании параллельных алгоритмов рассматриваются размещения матриц (двумерных массивов) «по блокам», «по полосам строк», «по полосам столбцов», «по скошенным диагоналям». Эти описания могут быть описаны как блочно-аффинные размещения по модулю количества ПЭ.

Определение 1. [5, 6] *Пусть натуральные (включая ноль) числа* $d1, d2, \ldots, dm$ *и целые константы* $s0, s1, s2, \ldots, sm$ *зависят только от* m*-мерного массива* X*. Блочно-аффинное по модулю* p *размещение* m*-мерного массива* X *– это такое размещение, при котором элемент* $X[i1, i2, \ldots, im]$ *находится в модуле памяти с номером* $u = (ceil(s1/d1) * i1 + ceil(s2/d2) * i2 + \ldots + ceil(sm/dm) * im + s0) \bmod p$ *(Здесь и далее* $ceil(a)$ *– наименьшее целое, которое не меньше* a*).*

Число $s0$ показывает номер модуля памяти, в котором размещается нулевой элемент $X(0, 0, \ldots, 0)$. При блочно-аффинном способе размещения m-мерный массив представляется как массив блоков размерности $d1 * d2 * \ldots * dm$, который размещается так, что у каждого блока все элементы оказываются в одном модуле памяти.

Числа $p, d1, d2, \ldots, dm, s0, s1, s2, \ldots, sm$ будем называть параметрами размещения. Параметры $s1, s2, \ldots, sm$ обычно равны 0 или 1, в редких случаях бывает -1 [5].

3. Распараллеливание для ВС с распределенной памятью

Будем использовать запись Par_For для оператора цикла, разные итерации которого выполняются одновременно асинхронно на разных вычислительных устройствах (узлах кластера или ядрах процессора). Реализация такого оператора возможна в MPI и в OpenMP. Может возникнуть ситуация, когда некоторые данные в одном месте тела цикла LoopBody должны быть размещены одним способом, а в другом месте – другим. Тогда такие данные нужно либо дублировать, либо переразмещать (пересылать). Будем рассматривать циклические пересылки Par_Transfer(k), у которых из каждого ПЭ с номером $s = 0, 1, \ldots, (p-1)$ пересылается данное в ПЭ с номером $(s + k) \bmod p$ (здесь p – количество ПЭ).

Пример 1. Рассмотрим циклическую пересылку элементов массива B в массив A, находящийся в соседнем слева процессорном элементе: Par_Transfer(-1): (j = 1; j <= p; j = j+1) A[j] <- B[j]

Запись корректна, если эти элементы находятся в разных ПЭ, причем номер ПЭ, в котором $A[j]$ на 1 меньше, чем номер ПЭ, в котором лежит $B[j]$ (пересылка циклическая). Для всех j пересылка элементов $B[j]$ может выполниться одновременно.

Рассмотрим распараллеливание цикла 1. Пусть в теле внешнего цикла 1 имеется одномерный массив A длины N. Чтобы равномерно загрузить p процессорных элементов ВС, нужно, чтобы в каждом ПЭ оказалось N/p элементов массива A. Рассмотрим блочно-аффинное размещение этого массива с параметрами $d1 = N/p$, $s1 = 1$ ($s0$ – в зависимости от индексных выражений).

В ПЭ с номером k должны быть элементы исходного массива A с номерами от $k*(N/p)$ до $(k+1)*(N/p)-1$, $k = 0, 1, \ldots, (p-1)$. В некоторых параллельных алгоритмах в каждом ПЭ кроме элементов массива из указанного диапазона необходимо иметь несколько элементов этого же массива из соседних ПЭ. Количество элементов из соседнего ПЭ слева и количество элементов из соседнего ПЭ справа будем считать одинаковыми (для простоты изложения) и обозначим это число через M. Для этих элементов в каждом ПЭ отводятся некоторые области памяти (для M чисел каждая), которые обозначим соответственно LeftShadow[k] и RightShadow[k] $k = 0, 1, \ldots, (p-1)$. При $M = 1$ псевдокод с пересылками данных может выглядеть следующим образом:

Листинг 4. Псевдокод с пересылками

```
Par_Transfer(-1): (j = 1; j < = p; j = j+1) {
    LeftShadow[j-1] <- A[p*j+1]
}
Par_Transfer(-1): (j = 1; j < = p; j = j+1) {
    RightShadow[j+1] <- A[p*j-1]
}
Par_For (j1 = 1; j1 < = p; j1 = j1+1) {
    For (j2 = 1; j2 < = N/p; j2 = j2+1) {
        j = (N/p-1)*j1+j2;
        LoopBody(j)
    }
}
```

4. Размещения массивов с перекрытиями в распределенной памяти

При размещении с перекрытиями дополнительная память для элементов некоторого массива выделяется не в дополнительных массивах, а в этом же массиве (при этом массив увеличивается). В работах [10, 11] рассмотрены размещения массивов с (кратными) перекрытиями, которые дают ускорение при выполнении многих итерационных алгоритмов за счет уменьшения количества итераций. Размещения с перекрытиями оказываются удобными при организации пересылок данных.

Пример 2. Пусть задан массив $a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]$. Пусть $N = 12$, $p = 3$, кратность перекрытия $M = 1$.

- Разместим массив в ПЭ по $d = ceil(N/p)$

$$[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12].$$

- Преобразуем блоки так, чтобы каждый хранил M граничных элементов соседа:

$$[1, 2, 3, 4, 5], [4, 5, 6, 7, 8, 9], [8, 9, 10, 11, 12].$$

- Получим результирующий массив [1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9, 8, 9, 10, 11, 12] длины 16. Можно дополнить массив в начале и в конце элементами 0 (обычно эти элементы имеют значения краевых условий) для однородности – получим массив длины 18:

$$b = [0, 1, 2, 3, 4, 5], [4, 5, 6, 7, 8, 9], [8, 9, 10, 11, 12, 0]. \quad (1)$$

Размещение (1) – это блочно-аффинное размещение с параметрами: $N = 18$, $p = 3$, кратность перекрытия $M = 0$ (т.е. без перекрытия).

Чтобы не заводить отдельные массивы LeftShadow и RightShadow, дополним исходный массив A новыми элементами, содержащими те данные, которые должны быть в LeftShadow и RightShadow. В ПЭ с номером k должны быть элементы исходного массива A с номерами от $k*(N/p)-M$ до $(k+1)*(N/p)-1+M$, $k = 0, \ldots, (p-1)$ (слева и справа добавлено по M элементов). Всего $(N/p + 2 * M)$ элементов. Будем рассматривать новый массив AA от 0 до $N + p * 2 * M - 1$, элементами которого являются элементы исходного массива A (некоторые элементы встречаются дважды). В каждом ПЭ находится $(N + p * 2 * M)/p = N/p + 2 * M$ элементов этого массива. В ПЭ с номером k находятся элементы массива AA с номерами от $(N/p + 2 * M) * k$ до $(N/p + 2 * M) * (k + 1) - 1$. Для индекса j элемента массива $AA[j]$ вычислим:

```
j1 = j div (N/p+2*M) // номер ПЭ, в котором элемент AA[j]
j2 = j mod (N/p+2*M) // локальный номер элемента AA[j] внутри ПЭ,
                        считая от нуля
```

Значение элементов массива AA определяются следующим образом:

```
j = j1*(N/p+2*M) + j2
AA[j] = A[j1*(N/p)+j2]
```

5. Численный алгоритм решения задачи теплопроводности

Для иллюстрации представленного в данной статье метода рассмотрим один из численных алгоритмов решения двумерной задачи теплопроводности.

Пример 3. Рассмотрим один из алгоритмов, численно решающий одномерную задачу теплопроводности.

Рассмотрим одномерное уравнение теплопроводности:

$$\frac{\partial u}{\partial t} = a^2 \cdot \frac{\partial^2 u}{\partial x^2}.$$

Для явной разностной схемы выберем достаточно малые t и h – шаги по временному и пространственному измерениям соответственно [14].

Листинг 5. Цикл для уравнения теплопроводности

```
C1 = sqr(a)/sqr(h)
C2 = 1/t-2*C1
C3 = C1

For (i = 1; i < = T; i = i+1) {
  For (j = 1; j < = N; j = j+1) {
    Y[j] = C1*X[j-1]+C2*X[j]+C3*X[j+1]}
  For (j = 1; j < = N; j = j+1) {
    X[j] = Y[j] }
}
```

В приведенном псевдокоде объемлющий цикл не допускает распараллеливания. Распараллеливание допускают внутренние циклы со счетчиками j.

6. Использование размещения массивов с перекрытиями для оптимизации параллельных вычислений на ВС с распределенной памятью

Рассмотрим задачу распараллеливания программы из примера 3. Чтобы выполнять этот код параллельно на p ПЭ, можно для внутренних циклов сделать преобразование «гнездование цикла» с количеством итераций внешнего цикла, равным p. Тогда количество итераций внутреннего цикла будет равно N/p (полагаем, что N делится нацело на p):

Листинг 6. Гнездование цикла

```
For (i = 1; i < = T; i = i+1) {
  For (j1 = 1; j1 < = p; j1 = j1+1) {
    For (j2 = 1; j2 < = N/p; j2 = j2+1) {
      j = (j1-1)*p + j2;
      Y[j] = C1*X[j-1]+C2*X[j]+C3*X[j+1]
    }
  }
  For (j1 = 1; j1 < = p; j1 = j1+1) {
    For (j2 = 1; j2 < = N/p; j2 = j2+1) {
      j = (j1-1)*p + j2;
      X[j] = Y[j]
    }
  }
}
```

Будем рассматривать параллельное выполнение на p ПЭ циклов со счетчиком $j1$. При этом циклы со счетчиком $j2$ будут выполняться последовательно в каждом ПЭ. Для такого параллельного выполнения следует разместить массивы X и Y так, чтобы в каждом ПЭ находилось большинство необходимых данных. Таким естественным размещением массивов Y и X является блочно-аффинное размещение с параметрами $d = N/p$, $s1 = 1$, $s0 = 0$. Для каждого $k = 1, 2, \ldots, (p-1)$ выполнения (последовательного) на границах первого цикла со счетчиком $j2$ вычисления элементов $Y[k*N/p-1]$ и $Y[k*N/p]$ требуют из соседнего ПЭ переслать элементы $X[k*N/p]$ и $X[k*N/p-1]$ соответственно. При выполнении циклов со счетчиком $j2$ первую и последнюю итерации следует вычислять отдельно, поскольку используются данные из другого ПЭ. Аналогично, для вычисления $X[k*N/p-1]$ и $X[k*N/p]$.

Заменим массивы X и Y массивами с перекрытиями XX и YY. Тогда фрагмент программы 6 примет вид:

Листинг 7. Размещение с перекрытиями

```
For (i = 1; i < = T; i = i+1) {
    Par_Transfer(-1): (j1 = 1; j1 < = p; j1 = j1+1) {
        XX[j1*(N/p+2)-1] <- XX[j1*N/p]
    }
    /* пересылка влево по одному элементу из каждого ПЭ */
    Par_Transfer(+1): (j1 = 1; j1 < = p; j1 = j1+1) {
        XX[j1*(N/p+2)] <- XX[N/p*j1-1]
```

```
    }
    /* пересылка вправо по одному элементу из каждого ПЭ */
    Par_For (j1 = 1; j1 < = p; j1 = j1+1){
        /* параллельное выполнение */
        For (j2 = 1; j2 < p+1; j2 = j2+1) {
            /* последовательное выполнение в каждом ПЭ */
            j = j1*(N/p-1) + j2
            YY[j] = (C1*XX[j-1]+C2*XX[j]+C3*XX[j+1])
        }
    }
    Par_Transfer(-1): (j1 = 1; j1 < = p; j1 = j1+1) {
        YY[j1*(N/p +2)-1] <- YY[N/p*j1]
    }
    /* пересылка влево по одному элементу из каждого ПЭ */
    Par_Transfer(+1): (j1 = 1; j1 < = p; j1 = j1+1) {
        YY[j1*(N/p +2)] <- YY[N/p*j1-1]
    }
    /* пересылка вправо по одному элементу из каждого ПЭ */
    Par_For (j1 = 1; j1 < = p; j1 = j1+1){
    /* параллельное выполнение */
        For (j2 = 1; j2 < = N/p; j2 = j2+1) {
        /* последовательное выполнение в каждом ПЭ */
            j = j1*(N/p-1) + j2;
            XX[j] = (C1*YY[j-1]+C2*YY[j]+C3*YY[j+1])/3
        }
    }
}
```

Пример 4. Рассмотрим абстрактную разностную схему.

Листинг 8. Абстрактная разностная схема

```
For (i = 1; i < = T; i = i+1) { ///////////////// итерации
    For (j = 2; j < = N; j = j+1) {
        Y[j] =  a*X[j-2] + b*X[j-1] + c*X[j] + d*X[j+1]+ e*X[i+2]
}
```

В таком случае переменной M следует присваивать значение $M = 2$.

Представленное преобразование массива к размещению с перекрытиями в распределенной памяти можно реализовывать автоматически с помощью компилятора. Можно такое преобразование реализовать в семействах компиляторов с открытым кодом GCC или LLVM. Но более интересно реализовать такое преобразование в source-to-source системах, таких как Rose-Compiler или OPC, поскольку, получая на выходе высокоуровневый код, можно его затем компилировать любым другим компилятором, даже с закрытым кодом, таким как ICC, PGI, MS-Compiler.

Заключение

Представленный метод допускает автоматизацию – реализацию в компиляторе в качестве оптимизирующего преобразования. Для многоядерных микросхем, ядра которых имеют локальную адресуемую память, предлагаемый в статье метод разме-

щения пересылаемых элементов в том же массиве, что и исходные элементы, позволяет читать эти элементы из оперативной памяти такими же кэш-линейками, что и элементы исходного массива. Кроме того, если бы пересылаемые из другого модуля памяти элементы лежали в отдельном массиве, читая аргументы вычислений во внутреннем цикле, пришлось бы использовать условные операторы. Таким образом, для многоядерных микросхем с распределенной памятью ядер размещения с перекрытиями минимизируют не только пересылки, но и обращения к оперативной памяти, а также количество вычислительных операций.

Данная работа – шаг на пути к созданию эффективных оптимизирующих компиляторов на параллельные вычислительные системы с распределенной памятью.

Исследование выполнено за счет гранта Российского научного фонда № 22-21-00671, https://rscf.ru/project/22-21-00671/.

Литература

1. Bondhugula, U. Automatic Distributed-Memory Parallelization and CodeGeneration using the Polyhedral Framework / U. Bondhugula // Technical Report, 2011. – URL: http://mcl.csa.iisc.ac.in/downloads/publications/uday11distmem-tr.pdf (дата обращения: 01.09.2022)
2. Bikonov, D. Three-Level Parallel Programming System for the Hybrid 21-Core Scalar-Vector Microprocessor NM6408MP / D. Bikonov, A. Puzikov, A. Sivtsov // Cybersecurity Issues. – 2019. – P. 22–34.
3. SoC Esperanto. – URL: https://www.esperanto.ai/technology (дата обращения: 01.09.2022)
4. Корнеев, В.В. Параллельное программирование / В.В. Корнеев // Программная инженерия. – 2022. – Т. 13, № 1. – С. 3–16.
5. Штейнберг, Б.Я. Оптимизация размещения данных в параллельной памяти / Б.Я. Штейнберг. – Ростов-на-Дону: Изд-во Южного федерального университета, 2010.
6. Штейнберг, Б.Я. Блочно-аффинные размещения данных в параллельной памяти / Б.Я. Штейнберг // Информационные технологии. – 2010. – № 6. – P. 36–41.
7. Bahtin, V.A. Solving Applied Problems Using DVM-system / V.A. Bahtin, D.A. Zaharov, A.S. Kolganov et al // Bulletin of the South Ural State University. Series: Computational Mathematics and Computer Science. – 2019. – V. 8, № 1. – P. 89–106.
8. Bahtin, V.A. Experience in Solving Applied Problems Using Irregular Grids Using DVM-System / V.A. Bahtin, D.A. Zaharov, A.A. Ermichev et al // Parallel Computing Echnologies. – 2018. – P. 241—252.
9. Krivosheev, N.M. Algorithm for Searching Minimum Inter-Node Data Transfers / N.M. Krivosheev, B. Steinberg // Computer Science. – 2021. – V. 193. – P. 306–313.
10. Ammaev, S. Combining Parallelization with Overlaps and Optimization of Cache Memory Usage / S. Ammaev, L. Gervich, B. Steinberg // Parallel Computing Technologies. – 2017. – P. 257–264.
11. Гервич, Л.Р. Автоматизация распараллеливания программ с блочным размещением данных / Л.Р. Гервич, Е.Н. Кравченко, Б.Я. Штейнберг и др. // Сибирский журнал вычислительной математики. – 2015. – Т. 18, № 1. – С. 41–53.
12. Zhangxiaowen Gong. An Empirical Study of the Effect of Source-Level Loop Transformations on Compiler Stability / Zhangxiaowen Gong, Zhi Chen, Justin Szaday et al // Proceedings of the ACM on Programming Languages. – 2018. –V. 2,№ OOPSLA. – P. 1–29.
13. Wolfe, M. More Iteration Space Tiling / M. Wolfe // Supercomputing. – Reno, 1989. – P. 655–664.

14. Самарский, А.А. Уравнения математической физики / А.А. Самарский, А.Н. Тихонов. – М.: Изд-во МГУ, 1999.

Лев Романович Гервич, ведущий программист, BroutonLab (г. Ростов-на-Дону, Российская Федерация), lgervith@gmail.com.

Борис Яковлевич Штейнберг, доктор технических наук наук, профессор, кафедра «Алгебра и дискретная математика», Южный федеральный университет (г. Ростов-на-Дону, Российская Федерация), borsteinb@mail.ru.

Поступила в редакцию 1 сентября 2022 г.

MSC 15A06 **DOI: 10.14529/mmp230105**

AUTOMATION OF THE APPLICATION OF DATA DISTRIBUTION WITH OVERLAPPING IN DISTRIBUTED MEMORY

***L.R. Gervich*[1], *B.Ya. Steinberg*[2]**
[1]BroutonLab, Rostov-on-Don, Russian Federation
[2]Southern Federal University, Rostov-on-Don, Russian Federation
E-mail: lgervith@gmail.com, borsteinb@mail.ru

The article deals with block-affine data layouts with overlapping for optimizing parallel computing in a distributed memory computing system. Examples of target computing systems are high-performance clusters and advanced systems on a chip with a large number of computing cores. It is proposed to describe the placement of an array with overlaps as a new array of slightly greater length, in which additional elements have the values of some elements of the original array. The possibility of developing an automatic transformation (by the compiler) of the usual allocation of an array in distributed memory into a new array containing overlaps is being considered. The proposed method is illustrated by a well-known numerical algorithm for solving the heat conduction problem.

Keywords: automation of parallelization; distributed memory; program transformations; data distribution; data transfer.

References

1. Bondhugula, U. Automatic Distributed-Memory Parallelization and CodeGeneration using the Polyhedral Framework. *Technical Report*, 2011. Available at: http://mcl.csa.iisc.ac.in/downloads/publications/uday11distmem-tr.pdf (accessed on 01.09.2022)
2. Bikonov D., Puzikov A., Sivtsov A. Three-Level Parallel Programming System for the Hybrid 21-Core Scalar-Vector Microprocessor NM6408MP. *Cybersecurity Issues*, 2019, pp. 22–34. DOI: 10.21681/2311-3456-2019-4-22-34
3. *SoC Esperanto*. Available at: https://www.esperanto.ai/technology (accessed on 01.09.2022)
4. Korneev V.V. Parallel Programming. *Software Engineering*, 2022, vol. 13, no. 1, pp. 3–16.
5. Shteynberg B.Ya. *Optimizacia razmeshhenija dannyh v parallel'noi pamyati* [Optimizing Data Placement in Parallel Memory]. Rostov-na-Donu, Southern Federal University Publishing, 2010. (in Russian)
6. Shteynberg B.Ya. Block-Affine Data Placements in a Parallel Memory. *Information Technologies*, 2010, no. 6, pp. 36–41.(in Russian)
7. Bahtin V.A., Zaharov D.A., Kolganov A.S. et al. Solving Applied Problems Using DVM-System. *Bulletin of the South Ural state University. Series: Computational Mathematics and Computer Science*, 2019, vol. 8, no. 1. pp. 89–106. (in Russian)

8. Bahtin V.A., Zaharov D.A., Ermichev A.A. et al. Experience in Solving Applied Problems Using Irregular Grids Using DVM-System. *Parallel Computing Technologies*, 2018, pp. 241–252.
9. Krivosheev N.M., Steinberg B. Algorithm for Searching Minimum Inter-Node Data Transfers. *Computer Science*, 2021, vol. 193, pp. 306–313.
10. Ammaev S., Gervich L., Steinberg B. Combining Parallelization with Overlaps and Optimization of Cache Memory Usage. *Parallel Computing Technologies*, 2017, pp. 257–264. DOI: 10.1007/978-3-319-62932-2-24.
11. Gervich L.R., Kravchenko E.N., Steinberg B.Y. et al. Automatic Program Parallelization with Block Data Distribution. *Numerical Analysis and Applications*, 2005, vol. 8, no. 1, pp. 35–45.
12. Zhangxiaowen Gong, Zhi Chen, Justin Szaday. An Empirical Study of the Effect of Source-Level Loop Transformations on Compiler Stability. *Proceedings of the ACM on Programming Languages*, 2018, vol. 2, no. OOPSLA, pp. 1–29.
13. Wolfe M. More Iteration Space Tiling. *Supercomputing*, Reno, 1989, pp. 655–664.
14. Tikhonov A.N., Samarskii A.A. *Equations of Mathematical Physics*. Oxford, Pergamon Press, 1963.

Received September 1, 2022