Автоматизация блочного размещения данных в оперативной памяти компилятором языка Си

М.В. Юрушкин

Южный федеральный университет

В данной статье приводится описание расширения языка Си, поддерживающего блочное размещение массивов, которое реализовано в системе ОРС (Оптимизирующая распараллеливающая система) в виде нескольких директив компилятора. В конце статьи приводятся результаты ускорения программ за счет использования реализованных директив.

1. Введение

Процесс оптимизации программы для работы на суперкомпьютере не имеет смысла без оптимизации работы этой программы на одном процессоре. Для многих задач одним из способов оптимизации доступа к памяти в рамках одного процессора является использование блочного размещения. Так, в статье [1] автором приводится описание алгоритма блочного умножения матриц, использующего двойное блочное размещение данных. По производительности реализованный алгоритм превосходит библиотеку MKL и библиотеку PLASMA. Стоит отметить, что библиотека PLASMA уже использует [2] блочное размещение в качестве основного.

Блочное размещение встречается в различных задачах. Так, в алгоритме генерации полигонов, используемом в методе конечных элементов, замена стандартного распределения на блочное дает ускорение в 1.7 раза [3]. В алгоритме умножения матриц библиотеки GotoBLAS блочное распределение матриц используется для реализации эффективной векторизации (register blocking) [4].

В то же время использование блочного размещения данных затруднено, т.к. требуется высокая квалификация программиста. Современные языки программирования используют стандартные схемы размещения данных в оперативной памяти, такие как размещение по строкам [5] (язык Си) и размещение по столбцам [5] (язык ФОРТРАН). Непосредственное применение блочного размещения увеличивает объем результирующей программы и порождает сложные индексные выражения во вхождениях блочно размещаемых массивов. Поэтому представляется интересной задача автоматизации поддержки блочного размещения массивов в языках программирования.

Такая автоматизация была реализована в системе ОРС (Оптимизирующая распараллеливающая система) в виде нескольких директив компилятора. В конце статьи приводятся результаты ускорения программ за счет данных директив на задачах линейной алгебры (LU-разложение матрицы, перемножение матриц), двумерной фильтрации изображений и т.д.

2. Блочное размещение данных

Блочное размещение – это способ хранения массива в памяти, при котором массив разбивается на блоки одинакового размера. Блоки матрицы хранятся в памяти последовательно без промежутков. Элементы, находящиеся внутри одного блока, хранятся в памяти стандартным образом, например по строкам.

В некоторых блочных алгоритмах блочное размещение массив дает существенное увеличение производительности. Рассмотрим блочный алгоритм умножения квадратных матриц C=A*B (рис. 1):

```
void matrix_mult(int N, int d) {
    ...

    double *A, *B, *C;
```

```
    A = malloc(sizeof(double)*N*N);
    B = malloc(sizeof(double)*N*N);
    C = malloc(sizeof(double)*N*N);

    //  инициализация матриц A, B
    …

    for(di = 0; di < N; i+=d)
    for(dj = 0; dj < N; j+=d)
    for(dk = 0; dk < N; k+=d)
        for(i = di; i < MIN(N, di+d); i++)
        for(k = dk; k < MIN(N, dk+d); k++)
        for(j = dj; j < MIN(N, dj+d); j++)
           C[i*N+j] += A[i*N+k]*B[k*N+j];
    …
}
```

Рис. 1. Блочный алгоритм умножения квадратных матриц

Если размер матриц N больше размера виртуальной страницы, то соседние по вертикали элементы матрицы будут находиться в различных виртуальных страницах. Нетрудно подобрать такой размер блока d, при котором TLB-кеш уже не будет способен хранить физические адреса всех используемых виртуальных страниц, и в программе будет происходить большое количество промахов к TLB-кешу. Ситуация усугубляется еще больше, т.к. в случае промаха к TLB-кешу процессор вынужден простаивать (в отличие от ситуации с промахом к кешу данных). Данную проблему можно решить, если разместить матрицы A,B,C блочно с размером блока равным d. В этом случае при перемножении блоков задействуется минимальное количество виртуальных страниц.

Если размер блока d не кратен размеру кеш-линейки, то блочное размещение увеличивает также эффективность использования кеша данных. Это связано с тем, что данные пересылаются кеш-линейками. Если размер блока не кратен размеру кеш-линейки, то при стандартном размещении данных в кеш данных будут попадать не только элементы перемножаемых блоков, но также элементы и соседних блоков. Кеш данных будет засоряться неиспользуемыми данными, что отрицательно скажется на производительности. Напротив, при блочном размещении в кеш будут подкачиваться только элементы перемножаемых блоков.

3. Директивы блочного размещения данных в компиляторе языка Си

В системе компиляторов ОРС поддержка блочного размещения массивов реализована в виде нескольких директив компиляции. Перед объявлением блочно размещаемого массива указывается директива.

#pragma ops array declare(name, array dimension size list, block dimension size list) (1)

Список параметров директивы:

- name – имя размещаемого массива
- array dimension size list – массив размерностей размещаемого массива по каждому измерению.
- block dimension size list – массив размерностей блока по каждому измерению.

В данной директиве указывается информация о размере массива, а также размере блоков, на которые его стоит разбить.

Оператор, в котором производится выделение памяти для массива A, следует пометить директивой

#pragma ops array allocate(name). (2)

Директива (2) заменяет исходный оператор выделения памяти на новый оператор освобождения памяти, в котором выделяется память под блочно размещаемый массив. Блочно размещаемый массив должен иметь размер кратный размеру блока, т.к. выполнение данного условия на практике сильно упрощает генерацию индексных выражений. Аналогично, оператор, в котором производится освобождение памяти массива A, следует пометить директивой

#pragma ops array release (name) (3)

Директива (3) заменяет исходный оператор освобождения памяти на новый оператор освобождения памяти, в котором выделяется память под блочно размещаемый массив. На Рис. 1. приведен пример использования таких директив в алгоритме умножения матриц.

```
void matrix_mult(int N, int d) {
    ...
    #pragma ops array declare(A, N, N, d, d)
    #pragma ops array declare(B, N, N, d, d)
    #pragma ops array declare(C, N, N, d, d)
    double *A, *B, *C;

    #pragma ops array allocate(A)
    A = malloc(sizeof(double)*N*N);
    #pragma ops array allocate(B)
    B = malloc(sizeof(double)*N*N);
    #pragma ops array allocate(C)
    C = malloc(sizeof(double)*N*N);

    //  инициализация матриц A, B
    …

    for(di = 0; di < N; i+=d)
    for(dj = 0; dj < N; j+=d)
    for(dk = 0; dk < N; k+=d)
        for(i = di; i < MIN(N, di+d); i++)
        for(k = dk; k < MIN(N, dk+d); k++)
        for(j = dj; j < MIN(N, dj+d); j++)
           C[i*N+j] += A[i*N+k]*B[k*N+j];

    #pragma ops array release(A)
    free(A);
    #pragma ops array release(B)
    free(B);
    #pragma ops array release(C)
    free(C);
}
```

Рис. 2. Блочный алгоритм умножения матриц с использованием директив блочного размещения матриц

Для того, чтобы компилятор переразметил данные, к входной программе предъявляется следующее требование: в программе не должно быть операций взятия адреса блочно размещаемого массива, кроме как в операциях выделения и освобождения памяти массива. Операции выделения и освобождения памяти должны быть аннотированы директивами (2) и (3) соответственно. Из данного ограничения следует в частности то, что

1. Блочно размещаемый массив нельзя передавать в качестве аргумента другим функциям.

2. Над элементами блочно размещаемого массива запрещается использовать векторные операции, которые поддерживаются я языке Си в виде встроенных функций.

В следующей главе приводятся результаты численных экспериментов, в которых используются описанные выше директивы блочного размещения данных.

4. Численные эксперименты

Для проверки эффективности и корректности реализованных директив был написан пакет прикладных блочных программ, аннотированных данными директивами. В таблице 1 приведены результаты сравнения производительности программ, скомпилированных с включенной опцией блочного размещения данных, а также без нее.

Таблица 1. Результаты тестирования директив блочного размещения в памяти

Название алгоритма	**Размер матриц**	**Размер блока**	**Время работы алгоритма без директив (сек)**	**Время работы алгоритма с директивами (сек)**	**Ускорение**
Блочное умножение квадратных матриц	2048x2048	256x256	14.93	11.2	25%
Блочное возведение матрицы в квадрат	2048x2048	256x256	81.37	17.36	78.6%
Блочное LU-разложение матрицы	2048x2048	256x256	27.4	14.44	47%
Блочное QR-разложение матрицы	2048x1024	256x256	19.6	17.11	12.7%
Двумерная свертка матрицы	1024x1024	256x256	17.9	10.54	41%

Результаты численных экспериментов подтверждают, что реализованные директивы могут существенно увеличить производительность результирующей программы. Кроме того, на данных тестовых примерах была проверена корректность реализованных директив.

5. Заключение

На данный момент, представляется интересным расширение класса входных программ, для которых блочное размещение имеет преимущество по сравнению со стандартными размещениями

6. Литература

1. 1.Л. Гервич, Б. Штейнберг, М. Юрушкин, Программирование экзафлопсных систем, "Открытые системы", № 08, 2013
2. 2.PLASMA Users' Guide, September 4th, 2010
3. 3.Neungsoo Park, Wenheng Liu, Viktor K. Prasanna, Cauligi Raghavendra. Efficient Matrix Multiplication Using Cache Conscious Data Layouts
4. 4.Kazushige Goto, Robert A. van de Geijn. Anatomy of High-Performance Matrix Multiplication. ACM Trans. Math. Software, Vol. 34, No. 3. (May 2008), pp. 1-25.
5. 5.S. Chatterjee, V.V. Jain, A.R. Lebeck, S. Mundhra, and M. Thottethodi, "Nonlinear Array Layouts for Hierarchical Memory Systems," Proc. 13th ACM Int'l Conf. Supercomputing, June 1999.