ОПТИМАЛЬНОЕ РАЗМЕЩЕНИЕ ДАННЫХ НА ЭТАПЕ КОМПИЛЯЦИИ

С.В. Полуян

Целью данной работы является создание алгоритма, позволяющего на этапе компиляции найти оптимальное размещение массивов с точки зрения загрузки данных в кэш-память. На основе этого алгоритма ведется разработка алгоритма оптимального размещения данных в параллельной памяти. Этот алгоритм программно реализован в Открытой распараллеливающей системе [1].

Под оптимальным размещением многомерных массивов в памяти компьютера будем понимать такое размещение, при котором происходит минимальное количество загрузок данных в кэш-память.

Оптимальное размещение многомерных массивов зависит от того, как эти массивы хранятся в памяти компьютера, и того, как они используются в программе [2,3].

В качестве способа изменения размещения данных будем использовать транспонирование массивов.

Определение: Пусть A – многомерный массив размерности m размера $N_1 \times N_2 \times \ldots \times N_p \times \ldots \times N_q \times \ldots \times N_m$. Тогда транспонированным массивом по индексам i_p и i_q называется такой многомерный массив B размерности m размера $N_1 \times N_2 \times \ldots \times N_q \times \ldots \times N_p \times \ldots \times N_m$, что

$$A[i_1, i_2,\ldots, i_p,\ldots, i_q,\ldots, i_m] = B[i_1, i_2,\ldots, i_q,\ldots, i_p,\ldots, i_m], \text{ где } i_k = 1,\ldots, N_k \qquad (1)$$

Операция транспонирования заключается в том, что индексы i_p и i_q в исходной матрице меняются местами [4].

Будем считать, что в памяти компьютера можно хранить только один вид массива (исходный или транспонированный по некоторому набору индексов).

Рассмотрим набор многомерных массивов $A_1,A_2,\ldots,A_m$. Пусть $\Sigma = (\sigma_1,\sigma_2,\ldots,\sigma_m)$ и $\Omega = (\omega_1,\omega_2,\ldots,\omega_m)$ - наборы перестановок индексов соответствующих массивов $A_1,A_2,\ldots,A_m$. Пусть C_Σ - количество заполнений кэш-линеек при обращении к массивам, полученным из $A_1,A_2,\ldots,A_m$, транспонированием в соответствии с перестановками $\sigma_1,\sigma_2,\ldots,\sigma_m$, а C_Ω – количество заполнений кэш-линеек при обращении к массивам, полученным из $A_1,A_2,\ldots,A_m$, транспонированием в соответствии с перестановками $\omega_1,\omega_2,\ldots,\omega_m$. Тогда массивы $A_1,A_2,\ldots,A_m$ лучше разместить в памяти компьютера транспонированными в соответствии с набором перестановок Σ, если $C_\Sigma < C_\Omega$.

Таким образом, для определения оптимального способа размещения многомерных массивов необходимо найти такой набор перестановок индексов массивов, чтобы количество загрузок кэш-линеек при использовании этих массивов было минимальным. Следовательно, задача оптимального размещения многомерных массивов сводится к нахождению такого набора перестановок Σ, что

$$C_\Sigma = \underset{\Omega \in \Delta}{Min}(C_\Omega) \qquad (2)$$

где Δ – множество всевозможных наборов перестановок индексов соответствующих массивов.

Рассмотрим алгоритм вычисления количества заполнений кэш-линеек. На вход алгоритму подается текст программы, для которого необходимо определить количество заполнений кэш-линеек в процессе выполнения этой программы.

Для моделирования доступа к кэш-памяти понадобится список заполнения кэш-строк, в котором будет содержаться информация о том, под какую область памяти занята та или иная строка.

Будем рассматривать кэш прямого отображения, в котором каждой ячейке оперативной памяти однозначно соответствует ячейка кэш-памяти по следующему правилу [5]:

$$NumLine = \frac{Addr}{CacheLineSize} \bmod \frac{CacheSize}{CacheLineSize} \qquad (3)$$

где NumLine – номер кэш-линейки, в которую попадут данные, расположенные по адресу Addr в оперативной памяти, CacheLineSize и CacheSize - длина кэш-линейки и размер кэш-памяти соответственно.

Таким образом, данные в кэше будут конфликтовать если выполняется следующее условие:

$$Addr(A[i_1][i_2]\ldots[i_m]) = Addr(B[j_1][j_2]\ldots[j_n]) + z*CacheSize, \text{ где } z \in Z \qquad (4)$$

Будем считать, что другие программы не оказывают влияния на кэш-память.

Описание алгоритма.

Сначала в списке заполнения кэш-строк помечаем все строки как пустые, а счетчику загрузок данных присваиваем ноль.

Для всех данных в соответствии с порядком их использования определяем номера кэш-строк, в которые они попадут в процессе выполнения программы. Если данные попадают в строку, помеченную в списке как пустую или заполненную другими данными, то перепомечаем эту строку и увеличиваем счетчик загрузок на единицу.

Конец алгоритма.

Таким образом, для работы алгоритма понадобится определить порядок доступа к данным и вычислить номера кэш-строк, в которые эти данные попадут.

Как уже было сказано выше, данные из оперативной памяти размещаются не в произвольном месте кэш-памяти, а в строго определенном. Номер кэш-строки, в которой окажется элемент массива, можно определить с помощью формулы (3). В этой формуле параметры CacheLineSize и CacheSize зависят от архитектуры вычислительной машины, а адрес произвольного элемента массива определяется относительно начала размещения массива и смещения.

Определить адрес элемента $A[i_1][i_2]\ldots[i_m]$ массива A размерности m размера $N_1 \times N_2 \times \ldots \times N_m$ можно с помощью одной из следующих формул:

$$Addr(A[i_1][i_2]...[i_m]) = Addr(A[0][0]...[0]) + \\ + \sum_{j=1}^{m}(i_j * \prod_{k=j+1}^{m} N_k) * SizeOf(TypeData)$$

- если массив хранится в памяти компьютера по строкам.

$$Addr(A[i_1][i_2]...[i_m]) = Addr(A[0][0]...[0]) + \\ + \sum_{j=1}^{m}(i_j * \prod_{k=1}^{j-1} N_k) * SizeOf(TypeData)$$

- если массив хранится в памяти компьютера по столбцам.

Эти формулы используются в зависимости от языка, на котором написана программа, так как массивы в разных языках программирования размещаются в памяти компьютера по-разному. Например, в языке Фортран массивы размещаются по столбцам, а в языках C и Pascal - по строкам [6].

Чтобы проанализировать порядок доступа к данным в процессе использования массивов, необходимо построить дерево обращений к элементам массивов, в узлах которого содержится информация о циклах, а в листьях находятся вхождения массивов в данный цикл.

По этому дереву можно определить, к каким элементам массивов происходит обращение, и, соответственно, подсчитать для них количество заполнений кэш-строк.

В качестве примера рассмотрим перемножение двух матриц B и C с некоторой модификацией матрицы C, и найдем для трех двумерных массивов A, B и C оптимальное размещение их в памяти компьютера при использовании в следующем фрагменте программы:

```
int const N = 100;
int main()
{
    int i,j,k;
    int A[N][N], B[N][N], C[N][N];

    for (i = 0; i < N; i=i+1)
        for (j = 0; j < N; j=j+1)
        {
            C[j][i] = 1;
            for (k = 0; k < N; k=k+1)
                A[i][j]= A[i][j] + B[i][k]*C[k][j];
        }
}
```

Для этого примера дерево обращений к элементам массива будет иметь следующий вид:

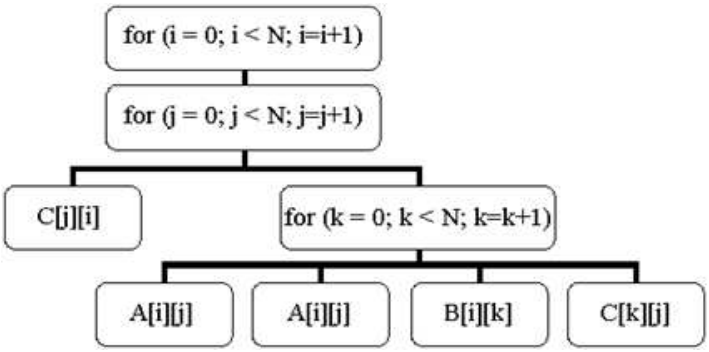

Рис.1. Дерево обращений к элементам массива

Рассмотрим все варианты размещения трех двумерных массивов в памяти компьютера, которые можно получить путем их транспонирования. Размер кэш-памяти целевой машины составляет 8 Кбайт. В результате работы алгоритма получаем следующие данные:

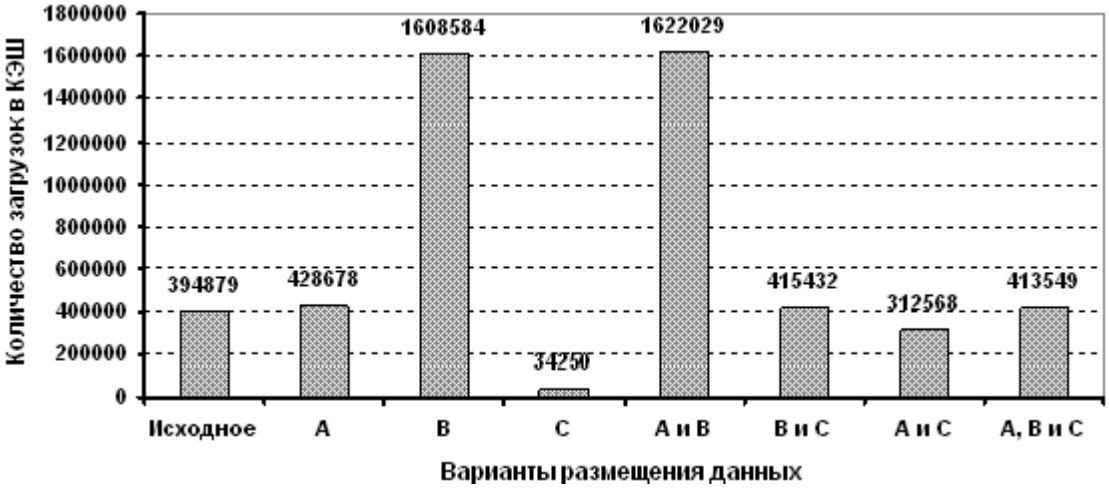

Рис.2. Результаты анализа заполнения кэш-линеек

где A, B и C – это массивы, которые были транспонированы для получения соответствующего способа размещения данных.

Теперь сравним полученные данные с реальным временем выполнения программ для всех вариантов размещения данных.

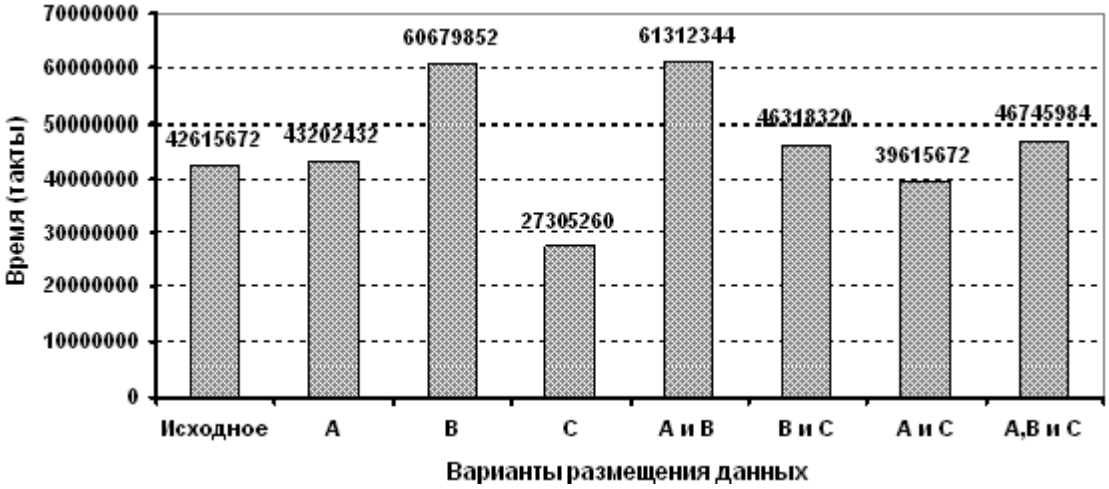

Рис.3. Время выполнения преобразованных программ

Таким образом, можно видеть, что результаты, полученные с помощью алгоритма определения оптимального размещения массивов, полностью подтверждаются данными, полученными путем измерения времени выполнения соответствующих программ.

ЛИТЕРАТУРА:

1. Открытая Распараллеливающая Система. URL: http://www.ops.rsu.ru
2. *T. M. Chilimbi, M. D. Hill, J. R. Larus* Making Pointer-Based Data Structures Cache Conscious – IEEE Computer, Vol. 33, No. 12, Dec. 2000, pp. 67-74.
3. *Ахо, Альфред В., Лам, Моника С., Сети, Равви, Ульман, Джеффри Д.* Компиляторы: принципы, технологии и инструментарий, 2-е изд.: Пер. с англ. – М.: ООО «И.Д. Вильямс», 2008. – 1184 с.: ил. – Парал. Тит. Англ.
4. *Ильин В.А., Позняк Э.Г.* Линейная алгебра. - М.: Наука, 1974.
5. Касперски К. Техника оптимизации программ. Эффективное использование памяти. – Спб.: БХВ-Петербург, 2003. – 464 с.: ил.
6. *Роберт У. Себеста.* Основные концепции языков программирования, 5-е изд.: Пер. с англ. — М.: Вильямс, 2001. — 672 с.