

Документ подписан простой электронной подписью

Информация о владельце:

ФИО: Локтионова Оксана Геннадьевна

Должность: проректор по учебной работе

Дата подписания: 03.02.2021 14:26:34

Уникальный программный ключ:

0b817ca911e6668abb13a5d426d39e5f1c11eabbf73e943df4a4851fda56d089

МИНОБРАЗОВАНИЯ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

Юго-Западный государственный университет
(ЮЗГУ)

Кафедра вычислительной техники

УТВЕРЖДАЮ

Проректор по учебной работе

Локтионова О.Г.

» 03.02.2021 г.



ВВЕДЕНИЕ В АЛГОРИТМИЧЕСКУЮ И ВЫСОКОУРОВНЕВУЮ ОПТИМИЗАЦИЮ

Методические указания по выполнению лабораторных и практических работ
для студентов направлений подготовки 09.03.01 и 09.04.01

Курск 2016

УДК 621.3

Составитель: Э.И. Ватутин

Рецензент

Кандидат технических наук, доцент *А.Г. Сневаков*

Введение в алгоритмическую и высокоуровневую оптимизацию: методические указания по выполнению лабораторных и практических работ по дисциплине «Параллельное программирование» / Юго-Зап. гос. ун-т; сост.: Э.И. Ватутин; Курск, 2016. 10 с.: ил. 2.

Методические рекомендации содержат сведения по разработке программных средств, направленных на уменьшение затрат времени выполнения программных реализацией алгоритма умножения плотных квадратных матриц на современных языках программирования высокого уровня в результате выполнения высокоуровневой и алгоритмической оптимизаций.

Предназначены для студентов направлений подготовки 09.03.01 и 09.04.01 «Информатика и вычислительная техника».

Текст печатается в авторской редакции

Подписано в печать _____ . Формат 60x84 1/16.

Усл. печ. л. Уч. – изд.л. Тираж 30 экз. Заказ . Бесплатно.

Юго-Западный государственный университет

305040, Курск, ул. 50 лет Октября, 94.

Содержание

Основные теоретические положения	4
Задание	7
Содержание отчета.....	8
Контрольные вопросы	9
Библиографический список.....	10

Основные теоретические положения

Цель работы: познакомиться с базовыми принципами выполнения алгоритмической и высокоуровневой оптимизации программного кода на примере задачи умножения плотных квадратных матриц.

Алгоритмической оптимизацией называется совокупность приемов, позволяющая решить ту же самую задачу с использованием более быстрых алгоритмов (реже, алгоритмов, реализация которых требует меньших затрат памяти, регистров и т.п.). Высокоуровневой оптимизацией называется оптимизация на уровне структурных элементов программы (подпрограмм, циклов, линейных участков). Обычно целью программной оптимизации является снижение времени, необходимого программе для решения поставленной задачи. В данной работе указанные виды оптимизации обсуждаются в контексте задачи умножения двух плотных квадратных матриц размера $N \times N$.

Умножение матриц «в лоб» реализуется в соответствии с известной формулой

$$c_{ij} = \sum_{k=1}^N a_{ik} b_{kj} \quad \forall i, j = \overline{1, N}$$

следующей программой (тип элементов матрицы – float):

```
for (int i=0; i<N; i++)
  for (int j=0; j<N; j++)
  {
    float s = 0.0f;

    for (int k=0; k<N; k++)
      s += A[i][k]*B[k][j];

    C[i][j] = s;
  }
```

Данная реализация довольно проста, однако неэффективна, когда размер матриц превышает объем кэш-памяти процессора. При выполнении умножения обход элементов матрицы A производится подряд

$(a_{i,1}, a_{i,2}, \dots, a_{i,N}, a_{i+1,1}, a_{i+1,2}, \dots, a_{i+1,N}, \dots)$, при этом эффективно работает кэш-память, предзагрузка данных реализуется с использованием механизма hardware prefetch и не требует внимания со стороны программиста. Обход элементов матрицы B напротив производится прыжками через $4N$ байт $(b_{1,j}, b_{2,j}, \dots, b_{N,j}, b_{1,j+1}, b_{2,j+1}, \dots, b_{N,j+1}, \dots)$, что мешает эффективной работе кэш-памяти и ограничивает быстродействие.

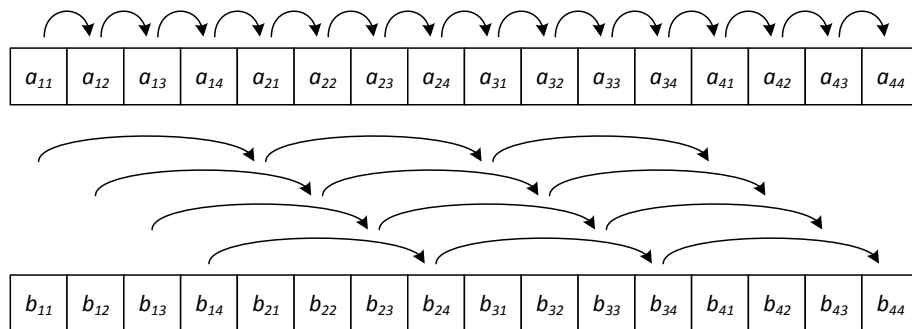


Рис. 1. Обращения к памяти при реализации умножения матриц «в лоб»

Чтобы обойти указанное ограничение, можно ввести временный массив

```
float tmp[N];
```

и занести в него элементы j -го столбца матрицы B . При этом при выполнении умножения i -х строк матрицы A , $i = \overline{1, N}$, на элементы j -го столбца кэш-память процессора работает максимально эффективно, однако при этом появляются накладные расходы, связанные с необходимостью копирования элементов столбца.

Еще одним способом умножения матриц является блочное умножение. Можно заметить, что результат умножения может быть получен путем независимого вычисления подматриц C'_{xy} матрицы C размером $S \times S$

элементов, $S = \frac{N}{k}$, k – обычно некоторое целое число (см. рис. 2).

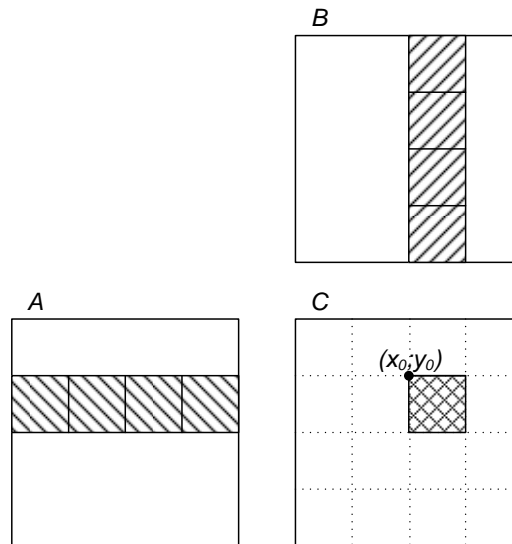


Рис. 2. Иллюстрация в блочному умножению матриц

При этом для вычисления значений, входящих в состав текущей подматрицы C'_{xy} , необходимы не все значения матриц A и B , а лишь узкие «полосы» значений размером $4 \times S \times N$ байт каждая, отмеченные на рисунке штриховкой. Дополнительно снизить требования к объему используемой кэш-памяти можно, заметив, что результирующие значения могут быть получены как суммы результирующих значений умножения подматриц. Для приведенного примера

$$C'_{23} = A'_{21} \times B'_{13} + A'_{22} \times B'_{23} + A'_{23} \times B'_{33} + A'_{24} \times B'_{43}.$$

Для вычисления каждого из произведений $A'_{xz} \times B'_{zy}$ необходимы лишь значения подматриц A'_{xz} и B'_{zy} объемом $4 \times S^2$ байт каждая.

При выполнении умножения основным действием в цикле является накапливающее сложение произведений элементов матриц:

```
for (int k=0; k<N; k++)
    s += A[i][k]*B[k][j];
```

Оно состоит из двух обращений к памяти, умножения и сложения. Обращения к памяти при чтении значений a_{ik} и b_{kj} теоретически могут производиться параллельно, в то время как умножение и последующее

сложение имеют последовательные зависимости по данным и могут выполняться только последовательно, что уменьшает параллелизм на уровне команд (англ. Instruction Level Parallelism, ILP). Для его увеличения необходимо осуществить раскрутку цикла (англ. Loop unrolling) на некоторое количество итераций M . Пример раскрутки цикла на 2 итерации приведен ниже:

```
for (int k=0; k<N; k+=2)
{
    s1 += A[i][k]*B[k][j];
    s2 += A[i][k+1]*B[k+1][j];
}
s = s1 + s2;
```

Раскрутка на разумное число итераций приводит к повышению ILP и, соответственно, к сокращению времени выполнения умножения матриц. Чрезмерная раскрутка цикла может привести к исчерпанию емкости кэша команд L1 и существенному увеличению времени обработки.

Задание

В соответствии с индивидуальным вариантом реализовать следующие варианты кода для умножения матриц размером 2048×2048 :

1. Классическое умножение «в лоб».
2. Умножение матриц с использованием буферизации столбца матрицы B .
3. Блочное умножение матриц (размер блока задается как параметр).

Сравнить быстродействие умножения «в лоб» в конфигурациях запуска «Debug» без оптимизации компилятора и «Release» с оптимизацией. Сделать выводы о получаемом выигрыше в скорости обработки.

Убедиться в том, что результаты умножения для всех алгоритмов совпадают. Для этого разработать отдельную подпрограмму, сравнивающую полученные матрицы C .

Измерить время выполнения блочного умножения t в зависимости от размера блока S (размер блока взять равным степеням двойки: 1, 2, 4, 8, ...), построить графики зависимости t от S для фиксированного N . Сделать вывод об оптимальном размере блока.

Произвести раскрутку внутренних циклов для умножения с буферизацией столбца и блочного умножения на различное число итераций M (выбирать равным степеням двойки, следить за размером блока). Оценить его влияние на время выполнения умножения, построить графики зависимости t от M , сделать вывод об оптимальном значении M для каждой из реализаций (для каждого значения S).

Сравнить быстродействие трех наилучших реализаций (с оптимальными значениями S и M) в конфигурации запуска «Release» для различных значений N (например, 4, 8, 16, 32, ..., 2048), сделать выводы о применимости на практике различных алгоритмов умножения матриц. Определить реальную производительность P процессора (в GFLOP/s) при выполнении умножений матриц с использованием разработанных программных реализаций, считая, что во внутреннем цикле производится 2 операции с плавающей точкой (floating point operation, FLOP).

Замеры времени и оценку производительности выполнить на 2–3 процессорах с различной архитектурой.

Содержание отчета

1. Титульный лист.
2. Цель работы.
3. Описание условия решаемой задачи в соответствии с индивидуальным вариантом.
4. Листинг программы.
5. Результаты сопоставления реализации «в лоб» в конфигурациях запуска «Debug» и «Release».

6. Результаты измерения времени умножения с использованием буферизации столбца для выбранного N , графики зависимости времени от M , выводы об оптимальном значении M .
7. Результаты измерения времени с использованием блочного умножения для выбранного N , графики зависимости времени от M и S , выводы об оптимальных значениях M и S .
8. Графики зависимости реальной производительности P от размера матриц N для 3 наилучших реализаций (выбранный алгоритм, оптимальная степень раскрутки M , оптимальный размер блока S), разработанных выше.
9. Выводы о применимости различных методов умножения матриц на практике.

Контрольные вопросы

1. Что называется алгоритмической оптимизацией?
2. Какие оптимизации относятся к высокоуровневым?
3. Что является ключевым недостатком базового алгоритма умножения матриц?
4. За счет чего буферизованное умножение оказывается быстрее базового алгоритма?
5. Сколько операций с плавающей точкой необходимо выполнить при умножении двух плотных квадратных матриц размера $N \times N$?

Библиографический список

1. Емельянов С.Г., Ватутин Э.И., Панищев В.С., Титов В.С. Процедурно-модульное программирование на Delphi: учебное пособие. М.: Аргмак-Медиа, 2014. 352 с.
2. Зотов И.В., Ватутин Э.И., Борзов Д.Б. Процедурно-ориентированное программирование на C++: учебное пособие. Курск: КурскГТУ, 2008. 211 с.