Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное   
учреждение высшего образования

Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет по лабораторной работе**

**«Умножение плотных матриц. Элементы типа double. Блочная схема, алгоритм Кэннона»**

**Выполнил**:

студент группы 381708-2

Пальгуев И.О.

**Проверил**:

Волокитин В.Д.

Нижний Новгород

2019

**Содержание**

[Введение 3](#_Toc33118001)

[Постановка задачи 4](#_Toc33118002)

[Метод решения 5](#_Toc33118003)

[Схема распараллеливания 7](#_Toc33118004)

[Описание программной реализации 7](#_Toc33118005)

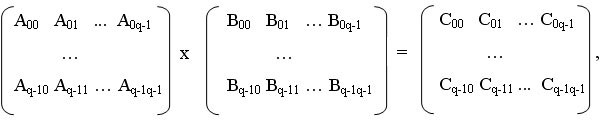
[Результаты экспериментов 9](#_Toc33118006)

[Заключение 9](#_Toc33118007)

[Список литературы 10](#_Toc33118008)

**Введение**

При блочной схеме разбиения матриц исходные матрицы А, В и результирующая матрица С представляются в виде наборов блоков. Будем предполагать далее, что все матрицы являются квадратными размера n\*n, количество блоков по горизонтали и вертикали являются одинаковым и равным q (т.е. размер всех блоков равен k\*k, k=n/q). При таком представлении данных операция матричного умножения матриц А и B в блочном виде может быть представлена в виде:



Где каждый блок Cij матрицы С определяется в соответствии с выражением:

http://it.kgsu.ru/ParalAlg/images/ris46_2.jpg

    При блочном разбиении данных для определения базовых подзадач естественным представляется взять за основу вычисления, выполняемые над матричными блоками. С учетом сказанного определим базовую подзадачу как процедуру вычисления всех элементов одного из блоков матрицы С.

**Постановка Задачи**

1. Реализовать умножение плотных матриц (алгоритмом Кэннона).

2. Реализовать параллельный алгоритм умножения плотных матриц алгоритмом Кэннона.

3. Провести вычислительные эксперименты.

4. Сравнить время работы последовательного и параллельного алгоритмов.

**Метод решения**

Начальное расположение блоков в алгоритме Кэннона подбирается таким образом, чтобы располагаемые блоки в подзадачах могли бы быть перемножены без каких-либо дополнительных передач данных. Алгоритма Кэннона включает выполнение следующих операций передач данных:

− в каждую подзадачу (i,j) передаются блоки Aij, Bij;

− для каждой строки i решетки подзадач блоки матрицы A сдвигаются на (i-1) позиций влево;

− для каждого столбца j решетки подзадач блоки матрицы B сдвигаются на (j-1) позиций вверх.

В результате такого начального распределения в каждой базовой подзадаче будут располагаться элементы, которые могут быть перемножены без дополнительных операций передачи данных.

**Схема распараллеливания**

Множество имеющихся процессоров представляется в виде квадратной решетки и  
размещение базовых подзадач (i,j) осуществляется на процессорах Pi,j соответствующих узлов  
процессорной решетки. Необходимая структура сети передачи данных может быть обеспечена  
на физическом уровне при топологии вычислительной системы в виде решетки или полного графа.

**Описание программной реализации**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <math.h>**

**#include "mpi.h"**

**#define DEBUG 1**

**void MatrixMultiplyAgg(int n, double \*a, double \*b, double \*c);**

**int main(int argc, char \*argv[])**

**{**

**int i, j;**

**int n, nlocal;**

**double \*a, \*b, \*c;**

**int npes, dims[2], periods[2];**

**int myrank, my2drank, mycoords[2];**

**int shiftsource, shiftdest;**

**int rightrank, leftrank, downrank, uprank;**

**MPI\_Status status;**

**MPI\_Comm comm\_2d;**

**MPI\_Init(&argc, &argv);**

**MPI\_Comm\_size(MPI\_COMM\_WORLD, &npes);**

**MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myrank);**

**if (myrank == 0)**

**printf("%d processors\n", npes);**

**if (argc != 2)**

**{**

**if (myrank == 0)**

**printf("Usage: %s <the dimension of the matrix>\n", argv[0]);**

**MPI\_Finalize();**

**exit(0);**

**}**

**dims[0] = sqrt(npes);**

**dims[1] = npes / dims[0];**

**if (dims[0] != dims[1])**

**{**

**if (myrank == 0)**

**printf("The number of processes must be perfect square.\n");**

**MPI\_Finalize();**

**exit(0);**

**}**

**// logical array of size ndims specifying whether the grid is**

**// periodic (true) or not (false) in each dimension**

**periods[0] = periods[1] = 1;**

**MPI\_Cart\_create(MPI\_COMM\_WORLD, 2, dims, periods, 0, &comm\_2d);**

**MPI\_Comm\_rank(comm\_2d, &my2drank);**

**MPI\_Cart\_coords(comm\_2d, my2drank, 2, mycoords);**

**n = atoi(argv[1]); // n**

**nlocal = n / dims[0];**

**a = (double\*)malloc(nlocal\*nlocal \* sizeof(double));**

**b = (double \*)malloc(nlocal\*nlocal \* sizeof(double));**

**c = (double \*)malloc(nlocal\*nlocal \* sizeof(double));**

**if (DEBUG)**

**printf("%d: init matrix\n", myrank);**

**for (i = 0; i < nlocal\*nlocal; i++) {**

**a[i] = myrank;**

**b[i] = myrank;**

**c[i] = 0.0;**

**}**

**if (DEBUG)**

**printf("%d: done initing matrix\n", myrank);**

**MPI\_Barrier(MPI\_COMM\_WORLD);**

**if (DEBUG) {**

**printf("%d: x:%d,y:%d\n", myrank, mycoords[0], mycoords[1]);**

**}**

**MPI\_Cart\_shift(comm\_2d, 0, -mycoords[1], &shiftsource, &shiftdest);**

**if (DEBUG) {**

**printf("%d: dest:%d,source:%d\n", myrank, shiftdest, shiftsource);**

**}**

**MPI\_Sendrecv\_replace(a, nlocal\*nlocal, MPI\_DOUBLE, shiftdest, 1, shiftsource, 1, comm\_2d, &status);**

**MPI\_Barrier(comm\_2d);**

**MPI\_Cart\_shift(comm\_2d, 1, -mycoords[0], &shiftsource, &shiftdest);**

**if (DEBUG) {**

**printf("%d: dest:%d,source:%d\n", myrank, shiftdest, shiftsource);**

**}**

**MPI\_Sendrecv\_replace(b, nlocal\*nlocal, MPI\_DOUBLE, shiftdest, 1, shiftsource, 1, comm\_2d, &status);**

**if (DEBUG)**

**{**

**printf("%d: ready to start calculating\n", myrank);**

**}**

**MPI\_Barrier(comm\_2d);**

**MPI\_Cart\_shift(comm\_2d, 0, -1, &rightrank, &leftrank);**

**MPI\_Cart\_shift(comm\_2d, 1, -1, &downrank, &uprank);**

**if (DEBUG)**

**{**

**printf("%d: right:%d, left:%d, up:%d, down:%d\n",**

**myrank, rightrank, leftrank, uprank, downrank);**

**}**

**for (i = 0; i < dims[0]; i++)**

**{**

**MPI\_Barrier(comm\_2d);**

**MatrixMultiplyAgg(nlocal, a, b, c);**

**MPI\_Sendrecv\_replace(a, nlocal\*nlocal,**

**MPI\_DOUBLE, leftrank, 1, rightrank, 1, comm\_2d, &status);**

**MPI\_Sendrecv\_replace(b, nlocal\*nlocal, MPI\_DOUBLE, uprank, 1, downrank, 1, comm\_2d, &status);**

**}**

**MPI\_Barrier(comm\_2d);**

**MPI\_Cart\_shift(comm\_2d, 0, -mycoords[1], &shiftsource, &shiftdest);**

**MPI\_Sendrecv\_replace(a, nlocal\*nlocal, MPI\_DOUBLE, shiftdest, 1, shiftsource, 1, comm\_2d, &status);**

**MPI\_Barrier(comm\_2d);**

**MPI\_Cart\_shift(comm\_2d, 1, -mycoords[0], &shiftsource, &shiftdest);**

**MPI\_Sendrecv\_replace(b, nlocal\*nlocal, MPI\_DOUBLE, shiftdest, 1, shiftsource, 1, comm\_2d, &status);**

**MPI\_Barrier(comm\_2d);**

**MPI\_Comm\_free(&comm\_2d);**

**if (DEBUG)**

**printf("%d: finish calculating\n", myrank);**

**MPI\_Barrier(MPI\_COMM\_WORLD);**

**int rank = 0;**

**while (rank < npes) {**

**if (myrank == rank) {**

**printf("my rank: %d\n", myrank);**

**//printf("x:%d,y:%d\n", mycoords[0], mycoords[1]);**

**puts("Random Matrix A");**

**for (i = 0; i < nlocal; i++)**

**{**

**for (j = 0; j < nlocal; j++)**

**printf("%6.3f ", a[i\*nlocal + j]);**

**printf("\n");**

**}**

**puts("Random Matrix B");**

**for (i = 0; i < nlocal; i++)**

**{**

**for (j = 0; j < nlocal; j++)**

**printf("%6.3f ", b[i\*nlocal + j]);**

**printf("\n");**

**}**

**puts("Matrix C = A\*B");**

**for (i = 0; i < nlocal; i++)**

**{**

**for (j = 0; j < nlocal; j++)**

**printf("%6.3f ",**

**c[i\*nlocal + j]);**

**printf("\n");**

**}**

**free(a);**

**free(b);**

**free(c);**

**}**

**rank++;**

**MPI\_Barrier(MPI\_COMM\_WORLD);**

**}**

**MPI\_Finalize();**

**return 0;**

**}**

**void MatrixMultiplyAgg(int n, double \*a, double \*b, double \*c)**

**{**

**int i, j, k;**

**for (i = 0; i < n; i++)**

**for (j = 0; j < n; j++)**

**for (k = 0; k < n; k++)**

**c[i\*n + j] += a[i\*n + k] \* b[k\*n + j];**

**}**

**Результаты экспериментов**

Эксперименты проводились на ПК с следующей конфигурацией:

* Операционная система: Windows 10 Pro
* Процессор: Intel(R) Core(TM) i5-7200U CPU @ 2.50 GHz
* ОЗУ 8 гб
* Версия Visual Studio: 2017

Таблица 1. Результаты экспериментов

|  |  |  |  |
| --- | --- | --- | --- |
| Все запуски параллельной программы производились на 4х процессах | | | |
| Количество входных данных | Время выполнения последовательной программы (сек.) | Время выполнения параллельной программы (сек.) | Ускорение |
| 12 | 0.001 | 0.031 | 0.0322 |
| 120 | 0.057 | 0.179 | 0.3184 |
| 1200 | 35.73 | 28.669 | 1.2462 |
| 1600 | 89.013 | 49.324 | 1.8046 |

**Заключение**

В ходе работы были реализованы два алгоритма умножения плотных матриц (алгоритмом Кэннона):

последовательное умножение плотных матриц и параллельное умножение плотных матриц. Вычислительные эксперименты показали, что

последовательный алгоритм уступает в производительности параллельному алгоритму, так как работает на большем объеме данных. Этот факт делает параллельное умножения плотных матриц (алгоритмом Кэннона) более предпочтительным для использования в реальных вычислительных задачах.

**Список литературы**

1. <http://www.hpcc.unn.ru/mskurs/RUS/DOC/ppr08.pdf>
2. Параллельные методы матричного умножения

<http://it.kgsu.ru/ParalAlg/palg046.html>