Была поставлена задача – найти произведение двух квадратных матриц. При чем каждая из матриц хранится в виде массива. Матрица А – нижняя треугольная, В - симметричная.

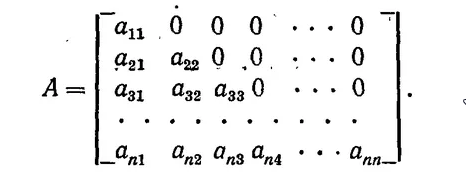


Рисунок 1 нижняя треугольная матрица

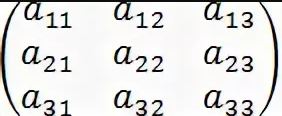
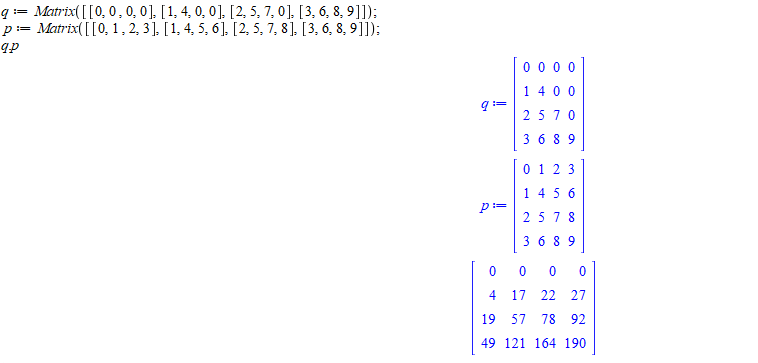


Рисунок 2 Симметричная матрица

В качестве блоков матрицы будем понимать квадратную подматрицу размера bSize. Для удобства размер блока будем брать равным , i=2..n/2.

Несколько слов о хранении матриц. Нижнетреугольная матрица предполагает, что выше главной диагонали лежат 0, следовательно, их хранить нецелесообразно. Такая же структура хранения и у матрицы В, только вместо 0, над главной диагональю лежат элементы симметричные элементам «нижнего треугольника» матрицы. Поэтому можно хранить лишь половину матрицы В.

Для простоты и экономии времени будем использовать один массив A=(0, 1, 2, …, m). m-число не нулевых элементов. При перемножении недостающие n\*n-m элементов матриц В и А будут восполняться за счет имеющихся элементов или 0.

Рассмотрим пример для n=4. Массив А = (0,1,2,3,4,5,6,7,8,9)  


Матрица А=q, B=p. Третья безымянная матрица – результат умножения А на В.

Теперь рассмотрим вывод консоли для размеров блока 1 и 2

Изображение выглядит как внешний

Описание создано автоматически

Как видим результаты умножения индентичны.

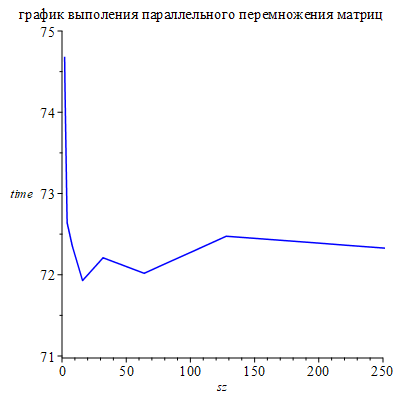
Теперь возьмем матрицы размера 1024 и определим оптимальное количество элементов в блоке для параллельного и непараллельного варианта реализации.

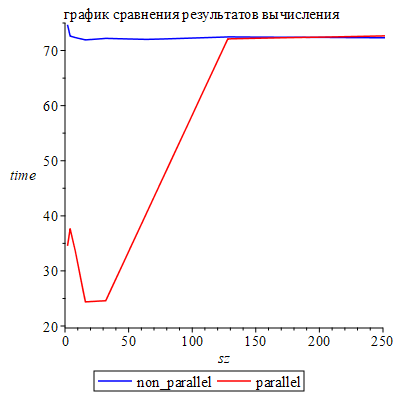
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| size | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 |
| Parallel (sec) | 34.5399 | 37.6258 | 33.6807 | 24.3871 | 24.5776 | 40.4564 | 72.1268 | 72.6727 |
| Non-parallel (sec) | 74.6838 | 72.6397 | 72.3552 | 71.9292 | 72.209 | 72.0178 | 72.4745 | 72.3222 |



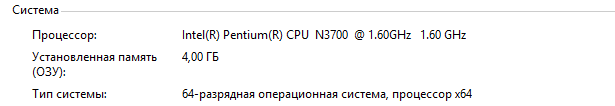
Наименьшее время выполнение у параллельного варианта было 24.3871, делаем вывод, что блок размера 16 оптимален. Отметим, что использовалось 4 потока.

У непараллельного варианта, очевидно, время будет в разы больше. Оптимальное 71.9292 и размер блока в 16 элементов.



Для наглядного сравнения двух реализаций: 

Характеристики компьютера:



Листинг

#include "pch.h"

#include <omp.h>

#include <iostream>

using namespace std;

int main()

{

int i, j, k, ik, jk, kk;

const int n = 512;

int bSize = 2;

int N = 0;

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

{

N += i;

}

int \*A = new int[N];

for (int i = 0; i < N; ++i)

{

A[i] = i;

}

double start\_time, end\_time;

for (bSize; bSize < n; bSize \*= 2)

{

int \*C=new int [n\*n];

cout << "size of matrix " << n << "\n" << "size of block " << bSize << "\n" << endl;

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

{

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

{

C[j \* n + i] = 0;

}

}

int a\_n = 0, b\_n = 0, g = 0, b=0, a=0;

start\_time = omp\_get\_wtime();

#pragma omp parallel private(i, j, k, ik, jk, kk, a, b, b\_n, a\_n, g)

{

#pragma omp for schedule(dynamic, 4)

for (jk = 0; jk < n; jk += bSize)

{

a = 0;

for (kk = 0; kk < n; kk += bSize)

{

a += kk;

for (ik = 0; ik < n; ik += bSize)

{

for (j = 0; j < bSize; j++)

{

for (k = 0; k < bSize; k++)

{

for (i = 0; i < bSize; i++)

{

if ((kk + k) < (ik + i)) { //если попадаем в элемент выше главной диагонали

break;

}

else

{

a\_n = (kk + k);

for (g = 1; g <= ik + i; ++g)

{

a\_n += n - g;

}

a = A[a\_n];

}

b = 0;

if (ik + i < jk + j) //если попадаем в элемент выше главной диагонали

{

b\_n = jk + j;

for ( g = 1; g <= ik + i; ++g)

b\_n += n - g;

b = A[b\_n];

}

else

{

b\_n = (ik + i);

for ( g = 1; g <= jk + j; ++g)

{

b\_n += n - g;

}

b = A[b\_n];

}

C[(kk + k) \* n + (jk + j)] += a \* b;

}

}

}

}

}

}

}

end\_time = omp\_get\_wtime();

cout << " time " << end\_time - start\_time << endl;

/\*for (int i = 0; i < n; ++i)

{

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

cout << C[i\*n + j] << " ";

cout << endl;

}\*/

cout << endl;

delete[]C;

}

for (bSize=2; bSize < n; bSize \*= 2)

{

int \*C = new int[n\*n];

cout << "size of matrix " << n << "\n" << "size of block " << bSize << "\n" << endl;

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

{

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

{

C[j \* n + i] = 0;

}

}

int a\_n = 0, b\_n = 0, g = 0, b = 0, a = 0;

start\_time = omp\_get\_wtime();

for (jk = 0; jk < n; jk += bSize)

{

a = 0;

for (kk = 0; kk < n; kk += bSize)

{

a += kk;

for (ik = 0; ik < n; ik += bSize)

{

for (j = 0; j < bSize; j++)

{

for (k = 0; k < bSize; k++)

{

for (i = 0; i < bSize; i++)

{

if ((kk + k) < (ik + i)) {

break;

}

else

{

a\_n = (kk + k);

for (g = 1; g <= ik + i; ++g)

{

a\_n += n - g;

}

a = A[a\_n];

}

b = 0;

if (ik + i < jk + j)

{

b\_n = jk + j;

for (g = 1; g <= ik + i; ++g)

b\_n += n - g;

b = A[b\_n];

}

else

{

b\_n = (ik + i);

for (g = 1; g <= jk + j; ++g)

{

b\_n += n - g;

}

b = A[b\_n];

}

C[(kk + k) \* n + (jk + j)] += a \* b;

}

}

}

}

}

}

end\_time = omp\_get\_wtime();

cout << " time " << end\_time - start\_time << endl;

delete[]C;

}

delete[] A;

}