МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Кафедра информационных систем

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1  
по курсу: Многопоточное программирование

Работу выполнила:

Лачинова Алиса

группа М4105

САНКТ-ПЕТЕРБУРГ

2015

**Задание:**

Реализовать программу по вычислению произведения матриц с использованием OpenMP. Исследовать эффективность распараллеливания и ускорения вычислений в зависимости от способов распараллеливания.

В качестве аргументов приложения должно принимать два файла с матрицами формата:

*x11 x12 x13*

*x21 x22 x21*

*x31 x32 x33*

В качестве результата -  файл с итогом вычислений. Время вычислений должно выводится на экран.

Итогом лабораторной работы является:

1. компилируемый программный код
2. отчет с исследованием эффективности распараллеливания в виде таблицы и технический характеристик используемого ПК.

**Результаты:**

Количество секунд измеряется с помощью метода clock(). Реализован класс матриц, в котором есть метод, осуществляющий перемножение матиц. Разница во времени измеряется между началом выполнения этого метода и завершением.

Ниже приведен код данного метода с разными способами распараллеливания:

1. Метод при последовательном выполнении

Matrix operator\*(const Matrix& vec\_other)

{

Matrix vec\_res;

vec\_res.vec.resize(this->vec.size());

if (vec\_other.vec.size() != this->vec[0].size())

{

cout << "Умножение не возможно. Размеры матриц не совпадают.";

return vec\_res;

}

for (int i = 0; i < this->vec.size(); i++)

{

vec\_res.vec[i].resize(vec\_other.vec[0].size());

for (int j = 0; j < vec\_other.vec[0].size(); j++)

{

double val = 0;

for (int z = 0; z < this->vec[0].size(); z++)

{

val += this->vec[i][z] \* vec\_other.vec[z][j];

}

vec\_res.vec[i][j]=val;

}

}

return vec\_res;

}

2. Метод при обычном распараллеливании.

Matrix operator\*(const Matrix& vec\_other)

{

Matrix vec\_res;

vec\_res.vec.resize(this->vec.size());

if (vec\_other.vec.size() != this->vec[0].size())

{

cout << "Умножение не возможно. Размеры матриц не совпадают.";

return vec\_res;

}

#pragma omp parallel

{

#pragma omp for

for (int i = 0; i < this->vec.size(); i++)

{

vec\_res.vec[i].resize(vec\_other.vec[0].size());

for (int j = 0; j < vec\_other.vec[0].size(); j++)

{

double val = 0;

for (int z = 0; z < this->vec[0].size(); z++)

{

val += this->vec[i][z] \* vec\_other.vec[z][j];

}

vec\_res.vec[i][j]=val;

}

}

}

return vec\_res;

}

3. Метод при распараллеливании с дополнительным условием

Так как после #pragma omp for распараллеливанию подлежит только первый цикл, а вложенные не учитываются при распалаллеливании. У нас 2 цикла, которые можно вложить один в другой 2мя способами. То есть for (int i = 0; i < this->vec.size(); i++) и for (int j = 0; j < vec\_other.vec[0].size(); j++). Лучше, чтобы у внешнего цикла было больше итераций. Тогда их можно продуктивнее распределять меду потоками.

Matrix operator\*(const Matrix& vec\_other)

{

Matrix vec\_res;

vec\_res.vec.resize(this->vec.size());

if (vec\_other.vec.size() != this->vec[0].size())

{

cout << "Умножение не возможно. Размеры матриц не совпадают.";

return vec\_res;

}

if (this->vec.size() >= vec\_other.vec[0].size())

{

#pragma omp parallel

{

#pragma omp for

for (int i = 0; i < this->vec.size(); i++)

{

vec\_res.vec[i].resize(vec\_other.vec[0].size());

for (int j = 0; j < vec\_other.vec[0].size(); j++)

{

double val = 0;

for (int z = 0; z < this->vec[0].size(); z++)

{

val += this->vec[i][z] \* vec\_other.vec[z][j];

}

vec\_res.vec[i][j] = val;

}

}

}

}

else

{

#pragma omp parallel

{

#pragma omp for

for (int i = 0; i < this->vec.size(); i++)

{

vec\_res.vec[i].resize(vec\_other.vec[0].size());

}

}

#pragma omp parallel

{

#pragma omp for

for (int j = 0; j < vec\_other.vec[0].size(); j++)

{

for (int i = 0; i < this->vec.size(); i++)

{

double val = 0;

for (int z = 0; z < this->vec[0].size(); z++)

{

val += this->vec[i][z] \* vec\_other.vec[z][j];

}

vec\_res.vec[i][j] = val;

}

}

}

}

return vec\_res;

}

4. Метод при распараллеливании с дополнительным условием dynamic

Matrix operator\*(const Matrix& vec\_other)

{

Matrix vec\_res;

vec\_res.vec.resize(this->vec.size());

if (vec\_other.vec.size() != this->vec[0].size())

{

cout << "Умножение не возможно. Размеры матриц не совпадают.";

return vec\_res;

}

if (this->vec.size() >= vec\_other.vec[0].size())

{

#pragma omp parallel

{

#pragma omp for ordered schedule(dynamic)

for (int i = 0; i < this->vec.size(); i++)

{

vec\_res.vec[i].resize(vec\_other.vec[0].size());

for (int j = 0; j < vec\_other.vec[0].size(); j++)

{

double val = 0;

for (int z = 0; z < this->vec[0].size(); z++)

{

val += this->vec[i][z] \* vec\_other.vec[z][j];

}

vec\_res.vec[i][j] = val;

}

}

}

}

else

{

#pragma omp parallel

{

#pragma omp for ordered schedule(dynamic)

for (int i = 0; i < this->vec.size(); i++)

{

vec\_res.vec[i].resize(vec\_other.vec[0].size());

}

}

#pragma omp parallel

{

#pragma omp for ordered schedule(dynamic)

for (int j = 0; j < vec\_other.vec[0].size(); j++)

{

for (int i = 0; i < this->vec.size(); i++)

{

double val = 0;

for (int z = 0; z < this->vec[0].size(); z++)

{

val += this->vec[i][z] \* vec\_other.vec[z][j];

}

vec\_res.vec[i][j] = val;

}

}

}

}

return vec\_res;

}

**Эффективность**:

|  |  |  |
| --- | --- | --- |
|  | матрица 2000x1000 и матрица 1000x4000 | матрица 4000x1000 и матрица 1000x2000 |
| Метод при последовательном выполнении | 2366.8 | 2355.03 |
| Метод при обычном распараллеливании dynamic,1 | 1010.48 | 1009.21 |
| Метод при распараллеливании с дополнительным условием dynamic ,1 | 998.447 | 1003.99 |
| Метод при обычном распараллеливании dynamic,10 | 1015.59 | 1010.72 |
| Метод при распараллеливании с дополнительным условием dynamic,10 | 1002.1 | 1008.09 |
| Метод при обычном распараллеливании guided,1 | 1009.7 | 1009.07 |
| Метод при распараллеливании с дополнительным условием guided,1 | 1009.64 | 1012.99 |
| Метод при обычном распараллеливании static,1 | 1021.88 | 1029.03 |
| Метод при распараллеливании с дополнительным условием static,1 | 1003.17 | 1008 |

Технические характеристики:

Процессор: Intel Core i5 2.5 GHz

ОЗУ: 4 Гб