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 и усовершенствованном алгоритме

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 и усовершенствованном алгоритме 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;

}

5. Метод при распараллеливании и усовершенствованном алгоритме dynamic,1

6. Метод при распараллеливании и усовершенствованном алгоритме dynamic,15

7. Метод при распараллеливании и усовершенствованном алгоритме dynamic,10

8. Метод при распараллеливании и усовершенствованном алгоритме guided,1

9. Метод при распараллеливании и усовершенствованном алгоритме guided,2

10. Метод при распараллеливании и усовершенствованном алгоритме static,1

**Эффективность**:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 2 матрицы 24x24 | матрица 24x34 и матрица 34x24 | матрица 34x24 и матрица 24x34 | матрица 24x24 и матрица 24x34 | матрица 34x24 и матрица 24x24 | матрица 49x24 и матрица 24x34 | матрица 34x24 и матрица 24x49 |
| Метод при последовательном выполнении | 0.006 | 0.007 | 0.011 | 0.008 | 0.01 | 0.015 | 0.018 |
| Метод при обычном распараллеливании | 0.003 | 0.005 | 0.008 | 0.007 | 0.008 | 0.012 | 0.014 |
| Метод при распараллеливании и усовершенствованном алгоритме | 0.003 | 0.005 | 0.008 | 0.007 | 0.007 | 0.012 | 0.012 |
| Метод при распараллеливании и усовершенствованном алгоритме dynamic | 0.003 | 0.005 | 0.007 | 0.006 | 0.007 | 0.008 | 0.008 |
| Метод при распараллеливании и усовершенствованном алгоритме dynamic,1 | 0.003 | 0.005 | 0.006 | 0.006 | 0.006 | 0.009 | 0.009 |
| Метод при распараллеливании и усовершенствованном алгоритме dynamic,15 | 0.005 | 0.007 | 0.008 | 0.007 | 0.007 | 0.009 | 0.009 |
| Метод при распараллеливании и усовершенствованном алгоритме dynamic,10 | 0.004 | 0.006 | 0.006 | 0.005 | 0.005 | 0.01 | 0.01 |
| Метод при распараллеливании и усовершенствованном алгоритме guided,1 | 0.003 | 0.004 | 0.005 | 0.005 | 0.005 | 0.008 | 0.008 |
| Метод при распараллеливании и усовершенствованном алгоритме guided,2 | 0.003 | 0.004 | 0.006 | 0.006 | 0.006 | 0.009 | 0.009 |
| Метод при распараллеливании и усовершенствованном алгоритме static,1 | 0.006 | 0.007 | 0.008 | 0.007 | 0.007 | 0.012 | 0.012 |

Технические характеристики:

Процессор: Intel Core i5 2.5 GHz

ОЗУ: 4 Гб