**Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования**

**"Уфимский университет науки и технологий"**

**Кафедра** Высокопроизводительных вычислительных технологий и систем

**Дисциплина:** Технологий параллельного программирования

**Отчет по лабораторной работе № 3**

**Тема:** «Параллельное вычисление произведения матриц с использованием OpenMP»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Группа ПМ-357 | Фамилия И.О. | Подпись | Дата | Оценка |
| Студент | Акмурзин М.Э. |  |  |  |
| Принял | Юлдашев А.В. |  |  |  |

**Уфа 2023**

**Цель работы:** приобрести навыки распараллеливания вложенных циклов с использованием директив OpenMP, исследовать ускорение, эффективность и производительность многопоточных реализаций алгоритмов решения задачи матричного умножения.

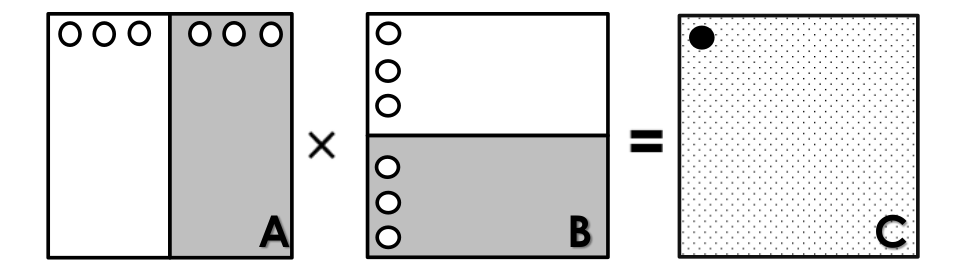
**Теоретический материал**

Послдатричное умножение выражается формулой:

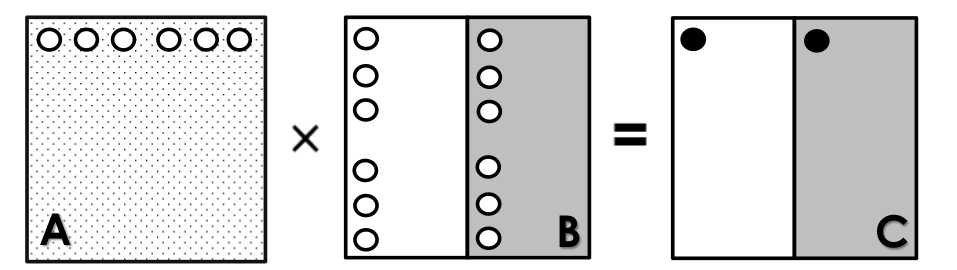
Квадрат евклидовой нормы:

Помимо последовательного алгоритма перемножения матриц, есть ещё 3 алгоритма параллельного перемножения матриц.

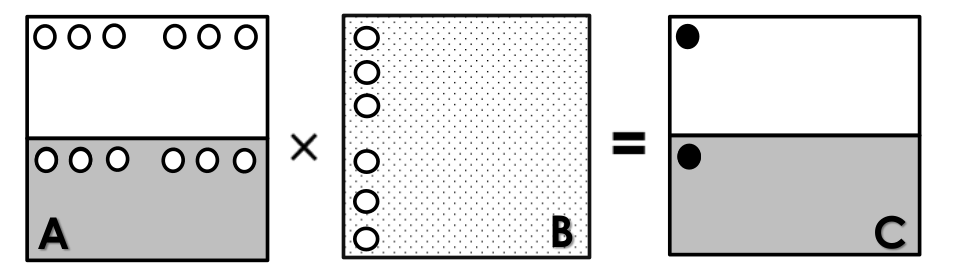
Алгоритм №1:



Алгоритм №2:



Алгоритм №3:



**Определение.** Отношение времени выполнения параллельной программы на одном процессоре (ядре) ко времени выполнения параллельной программы на **p** процессорах 𝑇𝑝 называется ускорением при использовании **p** процессоров:

**Определение.** Отношение ускорения к количеству процессоров **p** называется эффективностью при использовании **p** процессоров:

**Определение.** Пиковой (теоретической) производительностью называется максимальное количество команд или операций, которое может теоретически выполнять вычислительная система в единицу времени при условии постоянной и полной загрузки всех ее исполнительных устройств.

Пиковая производительность чаще всего измеряется в количестве выполняемых в секунду операций с плавающей точкой – Flops (Floating point operation per second). Формула для расчета пиковой производительности имеет вид:

где

– количество процессоров или ядер вычислительной системы;

– теоретическое количество операций с плавающей точкой, которое может выполнять процессор или ядро за 1 такт;

– тактовая частота, на которой работает процессор (ядро).

**Определение.** Реальной производительностью называется количество команд или операций, которое выполняет вычислительная система в единицу времени для конкретного алгоритма(программы).

Чтобы вычислить один элемент результирующей матрицы , необходимо выполнить скалярное умножения строки матрицы на столбец матрицы – требуется совершить операций умножения и столько же операций сложения.

Тогда для вычисления полной матрицы требуется вещественных операций. Тогда если за время осуществляется матричных умножений, получим, что реальную производительность можно выразить формулами:

Чтобы оценить эффективность использования вычислительной системы при выполнении последовательного и параллельного алгоритмов, введем коэффициенты:

**Практическая часть**

**Задание:**

1. Выполнить программную реализацию на языке C/C++ последовательного алгоритма умножения двух квадратных матриц размера . Предусмотреть:

1) ввод количества повторов умножения пользователем с клавиатуры;

2) статическое выделение памяти для хранения матриц;

3) заполнение матриц случайными вещественными числами в диапазоне от −0.5 до 0.5;

4) повторение процедуры умножения раз;

5) вывод на экран квадрата евклидовой нормы результирующей матрицы и времени работы программы.

Время замерять при помощи функции .

2. Выполнить распараллеливание матричного умножения путем добавления директив в текст последовательной программы. Реализовать три алгоритма параллельного умножения, предложенные ранее.

3. Отладить написанные программы при небольших размерностях матриц на многоядерной вычислительной системе. Убедиться, что результат работы параллельных программ совпадает с полученным в последовательной версии и является стабильным при многократных запусках.

4. Запустить все три параллельные версии при размерности матриц и количестве повторов таком, что время работы каждой из программ составляло бы порядка 10 секунд. Выбрать наиболее производительную версию для дальнейшей работы.

5. Запустить последовательную программу при различных размерностях матриц , выбирая так, чтобы время работы программы составляло не менее 10 секунд. Запустить параллельную программу при аналогичных значениях и . Время работы каждой программы занести в таблицу.

6. Вычислить ускорение и эффективность параллельной программы для каждого , полученные значения занести в таблицу.

7. Построить графики зависимости ускорения и эффективности параллельной программы от размерности умножаемых матриц. Объяснить полученные результаты.

8. Определить пиковую производительность одного ядра и всей многоядерной системы , на которой производятся вычисления.

9. Вычислить количество вещественных операций, выполняемых при матричном умножении для каждого 𝑁, реальную производительность, достигнутую в последовательной () и параллельной () версиях для каждой размерности и отношения реальной производительности к пиковой и . Полученные значения занести в таблицу.

10. Построить график зависимости и от размерности умножаемых матриц. Объяснить полученные результаты.

11. Определить размерность матриц такую, что при любом параллельная версия программы работает медленнее, чем последовательная, а при имеется ускорение при использовании параллельной программы. Для определения провести дополнительные запуски обеих версий программ.

12. Модифицировать параллельную программу путем добавления спецификатора директивы так, чтобы порождение параллельной области происходило только при размерности матриц . Убедится в корректности работы полученной программы.

**Выполнение работы:**

Тактовая частота, на которой работает процессор (ядро) = 3,50ГГц;

Теоретическое количество операций с плавающей точкой, которое может выполнять процессор или ядро за 1 такт: 4 операций;

Количество ядер: 4.

Для выполнения задания 1 зафиксируем число повторов умножения , выполним умножения матриц размера , где зафиксируем .

Результат работы программы:

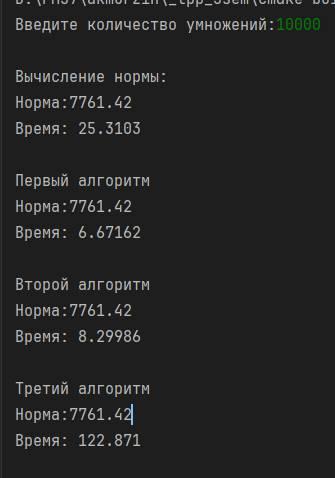


Рисунок 1 – Евклидова норма и время работы

Выполним распараллеливание матричного умножения путем добавления директив OpenMP в текст последовательной программы.

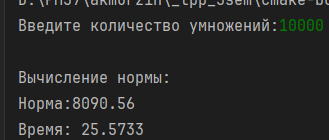


Рисунок 2 – Добавление директив OpenMP в код программы

Также реализуем три алгоритма параллельного умножения, предложенные ранее.

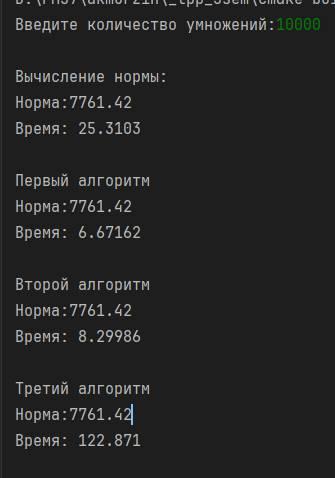


Рисунок 3 – Реализация трёх алгоритмов перемножения матриц

Из рисунка 3 видно, что результат работы параллельных программ совпадает с полученным в последовательной версии. Также результат является стабильным при многократных запусках. Время работы последовательной программы составляет секунд при .

На рисунке 3 показано, что наиболее производительной версией является алгоритм №1, поэтому в дальнейшем будем использовать его.

Запустим последовательную и параллельную программу при различных размерностях N, выбирая значение q так, чтобы время работы программы составляло не менее 10 секунд. Занесём результаты в таблицу 1.

Таблица 1 – время работы последовательной и параллельной программы

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 5 | 10000000 | 82.317 | 350.878 |
| 10 | 1000000 | 133.518 | 132.86 |
| 50 | 100000 | 180.846 | 367.697 |
| 100 | 10000 | 293.078 | 167.628 |
| 200 | 1000 | 126.781 | 87.06 |
| 500 | 60 | 98.733 | 60.9 |

Вычислим ускорение и эффективность параллельной программы для каждого 𝑁 по известным формулам, полученные значения занесём в таблицу 2.

Таблица 2 – ускорение и эффективность параллельной программы

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 5 | 0,030364 | 0,007591 |
| 10 | 0,314468 | 0,078617 |
| 50 | 1,716575 | 0,429144 |
| 100 | 2,376871 | 0,594218 |
| 200 | 2,625301 | 0,656325 |
| 500 | 2,30776 | 0,57694 |

Зависимость ускорения и эффективности параллельной программы от размерности умножаемых матриц весьма достаточно хорошо проиллюстрирована на графике 1 и графике 2.

График 1 – зависимость ускорения параллельной программы от размерности умножаемых матриц

График 2 – зависимость эффективности параллельной программы от размерности умножаемых матриц

Далее определим пиковую производительность одного ядра и всей многоядерной системы , на которой производятся вычисления:

Также вычислим количество вещественных операций, выполняемых при матричном умножении для каждого , реальную производительность, достигнутую в последовательной () и параллельной ( ) версиях для каждой размерности и отношения реальной производительности к пиковой   
 и .

Таблица 3 – полученные значения

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 5 | 3,2454888 | 0,985464 | 0,092201 | 0,027996 |
| 10 | 3,1567214 | 0,992688 | 0,08968 | 0,028201 |
| 50 | 1,9132793 | 3,284288 | 0,054355 | 0,093304 |
| 100 | 1,7005039 | 3,341878 | 0,04831 | 0,09494 |
| 200 | 1,2335934 | 3,560227 | 0,035045 | 0,101143 |
| 500 | 0,5036971 | 3,764424 | 0,01431 | 0,106944 |

Построим графики зависимости и от размерности умножаемых матриц.

График 4 – отношения реальной производительности к пиковой

Параллельная версия программы работает медленнее, чем последовательная при любом , где = 26.

**Вывод**

В ходе лабораторной работы были приобретены навыки распараллеливания вложенных циклов с использованием директив OpenMP, и исследовано ускорение, эффективность и производительность многопоточных реализаций алгоритмов решения задачи матричного умножения.

**Приложение**

#include <iostream>

#include <vector>

#include <omp.h>

#define N 100

double FirstMatrix[N][N];

double SecondMatrix[N][N];

double Product[N][N];

int main() {

int q = 0;

std::cout << "Please, enter the number of repetitions: ";

std::cin >> q;

std::cout << std::endl;

srand(time(NULL));

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

for (int j = 0; j < N; j++) {

FirstMatrix[i][j] = (rand() % 256) / 256.0 - 0.5;

SecondMatrix[i][j] = (rand() % 256) / 256.0 - 0.5;

}

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

//#pragma omp parallel for num\_threads(4)

for (int p = 0; p < q; p++) {

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

for (int j = 0; j < N; j++) {

Product[i][j] = 0;

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

Product[i][j] += FirstMatrix[i][k] \* SecondMatrix[k][j];

}

}

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

double Euclid = 0.0;

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

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

Euclid += Product[i][j] \* Product[i][j];

std::cout << "Square of euclidean norm:" << Euclid << std::endl;

std::cout << "Time: " << end\_time - start\_time << std::endl;

std::cout << std::endl;

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

for (int p = 0; p < q; p++) {

#pragma omp parallel for num\_threads(4)

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

for (int j = 0; j < N; j++) {

Product[i][j] = 0;

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

Product[i][j] += FirstMatrix[i][k] \* SecondMatrix[k][j];

}

}

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

Euclid = 0.0;

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

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

Euclid += Product[i][j] \* Product[i][j];

std::cout << "First algorithm" << std::endl;

std::cout << "Square of euclidean norm: " << Euclid << std::endl;

std::cout << "Time: " << end\_time - start\_time << std::endl;

std::cout << std::endl;

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

for (int l = 0; l < q; l++) {

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

#pragma omp parallel for num\_threads(4)

for (int j = 0; j < N; j++) {

Product[i][j] = 0;

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

Product[i][j] += FirstMatrix[i][k] \* SecondMatrix[k][j];

}

}

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

Euclid = 0.0;

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

for (int j = 0; j < N; j++) Euclid += Product[i][j] \* Product[i][j];

std::cout << "Second algorithm" << std::endl;

std::cout << "Square of euclidean norm: " << Euclid << std::endl;

std::cout << "Time: " << end\_time - start\_time << std::endl;

std::cout << std::endl;

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

for (int l = 0; l < q; l++) {

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

for (int j = 0; j < N; j++) {

Product[i][j] = 0;

double sum = 0.0;

#pragma omp parallel for reduction(+: sum) num\_threads(4)

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

sum += FirstMatrix[i][k] \* SecondMatrix[k][j];

Product[i][j] = sum;

}

}

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

Euclid = 0.0;

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

for (int j = 0; j < N; j++) Euclid += Product[i][j] \* Product[i][j];

std::cout << "Third algorithm" << std::endl;

std::cout << "Square of euclidean norm: " << Euclid << std::endl;

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

std::cout << "Time: " << end\_time - start\_time << std::endl;

std::cout << std::endl;

}