**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**"Уфимский государственный авиационный технический университет"**

**Кафедра** Высокопроизводительных вычислительных технологий и систем

**Дисциплина:** Технологии параллельного программирования

**Отчет по лабораторной работе № 3**

**Тема:** «Параллельное вычисление произведения матриц

с использованием OpenMP»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Группа МКН-315 | Фамилия И.О. | Подпись | Дата | Оценка |
| Студент | Гильманов И.И. |  |  |  |
| Принял | Спеле В.В. |  |  |  |

**Уфа 2022**

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

**Теоретический материал**

В состав Visual Studio включен компилятор языка C/C++ позволяющий создавать все, от простых консольных приложений, до универсальных приложений Windows, приложений Магазина Windows и компонентов .NET. Компиляция в консоли: cl [ключи] имя\_файла.c/cpp

Ключи оптимизации:

1. /Od – отключение оптимизаций (параметр по умолчанию в режиме Debug)
2. /O1 – максимальная оптимизация с приоритетом по уменьшению размера кода программы
3. /O2 – максимальная оптимизация с приоритетом по увеличению скорости работы программы (параметр по умолчанию в режиме Release)
4. /Ox – полная оптимизация
5. /Arch:[SSE|SSE2|AVX|AVX2|AVX512] – позволяет использовать векторные инструкции
6. /Qpar – автоматическое распараллеливание (происходит при выполнении определенных условий)

Компилятор Intel входит в состав Intel Parallel Studio (в среде Windows возможна интеграция в Visual Studio). Компиляция в консоли: icl [ключи] имя\_файла.c/cpp

Ключи оптимизации:

1. /Od (O0 в Linux) – отключение оптимизаций
2. /O1 – оптимизация по размеру
3. /O2 – максимизация скорости
4. /O3 – задействует оптимизации из /O2 и дополнительно более агрессивные методы оптимизации циклов и доступа к памяти,
5. /Ox – полная оптимизация
6. /QxHost - обеспечивает генерацию максимально современных векторных инструкций, поддерживаемых платформой /Qparallel – автоматическое распараллеливание (происходит при выполнении определенных условий)
7. /Qparallel – автоматическое распараллеливание (происходит при выполнении определенных условий)

Компилятор Intel входит в состав Intel Parallel Studio (в среде Windows возможна интеграция

Инструмент Intel Advisor XE является помощником разработчика многопоточных приложений. Он дает советы разработчику по использованию многопоточных технологий в приложении, автоматизируя анализ исходного кода, необходимый для быстрого и корректного внедрения многопоточных технологий в приложении.

OpenMP Application Program Interface - стандартизованный интерфейс разработки параллельных приложений для систем с общей памятью. Стандарт OpenMP определяет:

1. Директивы компилятора.
2. Функции библиотеки времени исполнения.
3. Переменные окружения.

Анализ полученных результатов распараллеливания:

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

*Определение*. Отношение ускорения к количеству процессоров называется эффективностью при использовании 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 (ядер) вычислительной системы; – теоретическое количество операций с плавающей точкой, которое может выполнять процессор (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 умножение 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:

**Индивидуальное задание**

Задание:

1. Выполнить программную реализацию последовательного алгоритма умножения двух квадратных матриц размера на языке C/C++. Предусмотреть:
2. ввод количества повторов умножения пользователем с клавиатуры;

2) заполнение матриц случайными вещественными числами в диапазоне от до ;

3) повторение процедуры умножения раз;

4) замер времени при помощи функции omp\_get\_wtime;

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

Использовать ранее изученные ключи оптимизации ипроанализировать производительность с помощью Intel Advisor.

1. Выполнить распараллеливание матричного умножения путем добавления директив OpenMP в текст последовательной программы. Реализовать три алгоритма параллельного умножения, предложенные ранее.
2. Отладить написанные программы при небольших размерностях матриц 𝑁 на многоядерной вычислительной системе. Убедиться, что результат работы параллельных программ совпадает с полученным в последовательной версии и является стабильным при многократных запусках. Проверить корректность с помощью Intel Inspector.
3. Запустить все три параллельные версии при размерности матриц и таком количестве повторов , при котором время работы каждой из программ составляло бы порядка секунд. Выбрать наиболее производительную версию для дальнейшей работы. Проанализировать эффективность распараллеливания с помощью Intel VTune Profiler.
4. Запустить последовательную программу при различных размерностях матриц , выбирая так, чтобы время работы программы составляло не менее секунд. Запустить параллельную программу при аналогичных значениях и . Время работы каждой программы занести в таблицу.
5. Вычислить ускорение и эффективность параллельной программы для каждого , полученные значения занести в таблицу.
6. Построить графики зависимости ускорения и эффективности параллельной программы от размерности умножаемых матриц. Объяснить полученные результаты.
7. Определить пиковую производительность одного ядра и всей многоядерной системы , на которой производятся вычисления.
8. Вычислить 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 . Полученные значения занести в таблицу.
9. Построить график зависимости и от размерности умножаемых матриц. Объяснить полученные результаты.
10. 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.
11. Модифицировать параллельную программу путем добавления спецификатора if директивы parallel for так, чтобы порождение параллельной области происходило только при размерности матриц . Убедиться в корректности работы полученной программы.

Результаты:

Тесты проводились на процессоре со следующими характеристиками:

A picture containing table

Description automatically generated

Рисунок 1. Характеристики процессора.

Были поставлены значения , . Последовательное вычисление происходит примерно за 16,8 сек.

После распараллеливания последовательного алгоритма для каждого из 3-ёх циклов было обнаружено, что результаты в последнем варианте (внутренний цикл) не сходятся с последовательной версией. Программа была проверена в Intel Inspector:

Text, table

Description automatically generated

Рисунок 2. Ошибка, найденная в Intel Inspector.

A picture containing graphical user interface

Description automatically generated

Рисунок 3. Ошибка в строчке, которую подсвечивает Inspector.

Для того чтобы избавиться от ошибки, необходимо в прагму для внутреннего цикла добавить reduction(+: sum).

Проверим программу в Intel Inspector ещё раз:

Text, letter

Description automatically generated

Рисунок 4. Сообщение об отсутствии проблем в программе.

Далее были запущены 3 параллельные версии при и таком количестве , при котором время работы программы составляло бы порядка 10 секунд.

Результаты тестирования:

Таблица 1. Результаты тестирования скорости 3 параллельных версий программы.

|  |  |  |  |
| --- | --- | --- | --- |
| Цикл / параметры |  |  |  |
| 1 (внешний) | 100 | 31000 | 10,1 |
| 2 | 100 | 24000 | 10,5 |
| 3 | 100 | 550 | 10,6 |

Из таблицы видно, что лучшая производительность достигается при использовании внешнего цикла, т.е. первой версии программы. Посмотрим эффективность распараллеливания в Intel VTune Profiler:

A picture containing chart

Description automatically generated

Рисунок 5. Эффективность распараллеливания в Intel VTune Profiler.

Далее была запущена последовательная программа при различных размерностях матриц , выбирая так, чтобы время работы программы составляло не менее 10 секунд, а также запущена параллельная программа при аналогичных значениях и . Для каждого было вычислено ускорение и эффективность паралелльной программы, определена пиковая производительность одного ядра и всей многоядерной системы , на которой производятся вычисления. Наконец, вычислено количество вещественных операций, выполняемых при матричном умножении для каждого , реальная производительность, достигнутая в последовательной () и параллельной () версиях для каждой размерности и отношения реальной производительности к пиковой и .

Все полученные данные занесены в таблицу:

Таблица 2. Результаты тестирования.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |
| 5 | 18000000 | 11.9 | 28.2 | 0.42 | 0.10 | 378151260 | 159348441 | 0,0028 | 0,0012 |
| 10 | 7000000 | 12 | 23.8 | 0.50 | 0.12 | 1166666667 | 588235294 | 0,0088 | 0,0044 |
| 50 | 90000 | 12.1 | 4.6 | 2.63 | 0.65 | 1859504132 | 4891304348 | 0,0141 | 0,0372 |
| 100 | 10000 | 11.4 | 3.4 | 3.35 | 0.83 | 1754385965 | 5882352941 | 0,0133 | 0,0448 |
| 200 | 1150 | 12.5 | 3.6 | 3.47 | 0.86 | 1472000000 | 5111111111 | 0,0112 | 0,0389 |
| 500 | 50 | 11 | 3 | 3.66 | 0.91 | 1136363636 | 4166666667 | 0,0086 | 0,0317 |

Построены графики зависимости ускорения и эффективности параллельной программы от размерности умножаемых матриц, а также график зависимости и от размерности умножаемых матриц.

Рисунок 6. Зависимость ускорения от размерности.

Рисунок 7. Зависимость эффективности ядер от размерности.

Рисунок 8. Зависимость эффективности использования вычислительной системы на одном ядре от размерности.

Рисунок 9. Зависимость эффективности использования вычислительной системы на всех ядрах от размерности.

Такое поведение связано с малой эффективностью распараллеливания программы на малом количестве вычислений в одной параллельной области, а также размерами кэшей и .

Далее была определена размерность матриц такая, что при любом параллельная версия программы работает медленнее, чем последовательная, а при имеется ускорение при использовании параллельной программы. Для определения проведены дополнительные запуски обеих версий программ.

Таблица 2. Результаты определения .

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 11 | 5000000 | 10.7 | 12.2 |
| 12 | 5000000 | 12.8 | 14.6 |
| 13 | 5000000 | 15.1 | 16.3 |
| 14 | 5000000 | 18 | 16.5 |

Параллельная программа была модифицирована путём добавления спецификатора if директивы parallel for так, чтобы порождение параллельной области происходило только при размерности матриц .

Text, letter

Description automatically generated

Рисунок 10. Проверка корректности работы модифицированной программы в Intel Inspector.

**Вывод**

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

**Приложение**

**main.cpp**

#include <iostream>

#include <vector>

#include <time.h>

#include <omp.h>

#define N 100

void Print(std::vector<std::vector<double>>& a) {

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

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

std::cout << a[i][j] << "\t";

std::cout << std::endl;

}

std::cout << std::endl;

}

void FillMatrix(std::vector<std::vector<double>>& a, std::vector<std::vector<double>>& b, std::vector<std::vector<double>>& c) {

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

a[i].resize(N);

b[i].resize(N);

c[i].resize(N);

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

a[i][j] = (double)rand() / RAND\_MAX - 0.5;

b[i][j] = (double)rand() / RAND\_MAX - 0.5;

}

}

}

std::vector<std::vector<double>> Multiply(std::vector<std::vector<double>>& a, std::vector<std::vector<double>>& b, std::vector<std::vector<double>>& c) {

#pragma omp parallel for if(N > 14)

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

//#pragma omp parallel for

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

double& sum = c[i][j];

//#pragma omp parallel for reduction (+: sum)

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

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

}

}

}

return c;

}

double Norma(std::vector<std::vector<double>>& a) {

double norm = 0;

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

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

norm += a[i][j] \* a[i][j];

}

}

return norm;

}

int main() {

omp\_set\_num\_threads(4);

std::vector<std::vector<double>> a(N), b(N), c(N);

srand(time(0));

int q = 100000; //10 000 000

//cout << "Input q = ";

//cin >> q;

FillMatrix(a, b, c);

double start = omp\_get\_wtime();

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

c = Multiply(a, b, c);

}

double end = omp\_get\_wtime();

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

c[i].clear();

c[i].resize(N);

}

c = Multiply(a, b, c);

double norm = Norma(c);

std::cout << "Time = " << end - start << std::endl;

std::cout << "Norma = " << norm << std::endl;

}