Министерство образования и науки Российской Федерации

Санкт–Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа киберфизических систем и управления

Отчет по лабораторной работе №4

Курс: практикум по программированию

Тема: многопоточность

Группа 23533/2

Студент:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Тарасов А.А.

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_Ковалевский В.Э

Санкт Петербург  
2019

# Многопоточность

Многопоточность – это свойство платформы или приложения, состоящее в том, что процесс в операционной системе может состоять из нескольких потоков, выполняющихся параллельно. Это позволяет эффективнее использовать машинный ресурс. Смысл многопоточности - квазимногозадачность на уровне одного исполняемого процесса. Выполняющийся процесс имеет как минимум один (главный) поток.

Многопоточность имеет как достоинства, так и недостатки:  
Преимущества – рост производительности при распараллеливании вычислений, перекладывание работы на более свободные ядра (в случае, если процессор не одноядерный), обеспечение плавности работы интерфейса или защита программ от непреднамеренных подвисаний (выделять на задачи программы по потоку, чтобы процесс не затрагивал основной поток)

Недостатки – возможная интерференция потоков, вопросы правильного распараллеливания, большое количество потоков может наоборот ухудшить работу системы.

Возможно решить проблему защиты данных, использующихся разными потоками. Для этого используется специальная концепция программирования – mutex – mutual exclusion – взаимное исключение. Мьютекс разрешает работать с некоторыми данными только одному потоку, пока он активен (заблокирован). В любое время данными обладает только тот поток, что закрыл мьютекс. Другой поток, оперирующий теми же данными, получит доступ к мьютексу только когда мьютекс разблокируется владеющим им потоком.

# Программа

Иллюстрируем пример программы, использующей многопоточность на примере программы, вычисляющей произведение матриц. Для упрощения кода, будем вычислять произведение 2х квадратных матриц размерностью SIZE. Пользователь вводит максимальное количество потоков, с которым он хочет вычислить произведение 2х матриц, затем произведение последовательно вычисляется для потоков 1 – Кол-во введённых пользователем. Процесс занимает время.

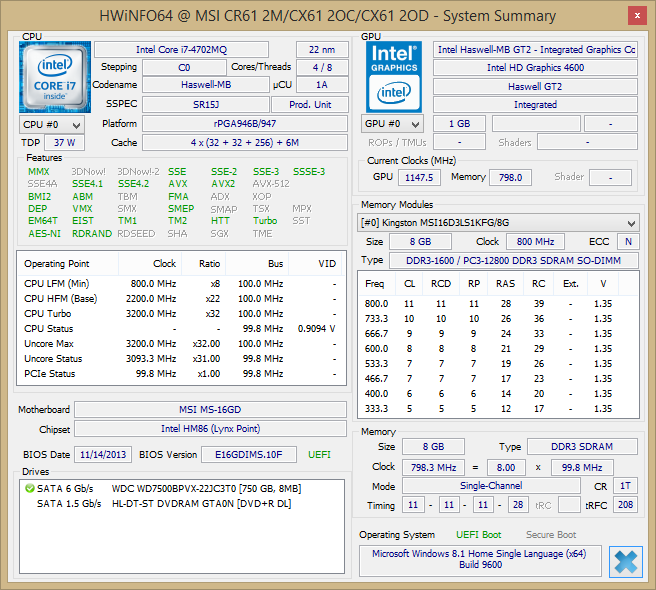
Здесь потоки создавались в runtime, а не при написании кода. Данные выгружаются в текстовый файл.

Программа разбивает матрицу С – результат перемножения – на несколько частей, соответственно количеству потоков. Каждые несколько рядов исходной матрицы высчитывает отдельный поток. На многоядерных процессорах это должно дать существенную прибавку к производительности, особенно на больших массивах данных. Здесь перемножались матрицы 1000х1000 элементов (int).

Характеристики машины приведены на рисунке ниже. Главное – процессор intel core i7 – 4702MQ, 4 ядра, гипертрединг, ~3 GHz при работе программы на ядро, 8 GB RAM DDR3.

Посмотрим, как библиотека thread, добавленная в с++11 и windows 8.1 справятся с разделением нагрузки на ядра. График для 1-8 потоков приведён ниже.

|  |  |  |
| --- | --- | --- |
| Кол-во потоков | | Время выполнения, мс |
| 1 | | 9607 |
| 2 | | 8234 |
| 3 | | 7807 |
| 4 | | 7711 |
| 5 | | 7634 |
| 6 | | 7445 |
| 7 | | 7404 |
| 8 | 7372 | |



Очень быстро время выполнения стремится вниз при использовании 1-4 потоков, дальше падение не столь быстрое. На больших матрицах потоки дадут большие выигрыши во времени. Например, выигрыш на матрицах 500х500 заметен гораздо хуже.

Исходный код:

// Multithreading.cpp: определяет точку входа для консольного приложения.

//

#include "stdafx.h"

#include <thread>

#include <chrono>

#include <iostream>

#include <ctime>

#include <fstream>

using namespace std;

using namespace std::chrono;

//Простое перемножение, по определению

void simple\_multiplication(int\*\* A, int\*\* B, int\*\* result, int size)

{

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

{

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

{//Пробег по каждой паре ij

result[i][j] = 0;

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

result[i][j] += A[i][k] \* B[k][j];

}

}

return;

}

void fill(int\*\* A, int size)

{

srand(time(0));

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

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

A[i][j] = rand() % 100;

}

void thread\_calculate(int\*\* A, int\*\* B, int\*\* result, int Lborder, int Rborder, int size)

{

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

{

for (int j = Lborder-1; j < Rborder; j++)

{//Пробег по каждой паре ij в пределах границы по столбцам

result[i][j] = 0;

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

result[i][j] += A[i][k] \* B[k][j];

}

}

return;

}

int main()

{

int SIZE = 1000;

//Каждый поток будет заполнять свои ряды в получившейся матрице.

int\*\* A = new int\*[SIZE];

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

A[i] = new int[SIZE];

int\*\* B = new int\*[SIZE];

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

B[i] = new int[SIZE];

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

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

C[i] = new int[SIZE];

fill(A, SIZE);

fill(B, SIZE);

int num\_threads = 0,j=0;

cout << "Enter maximum number of threads (takes time): ";

cin >> j;

//Тут прогнать несколько раз для кол-ва потоков

for (int num\_threads = 1; num\_threads <=j; num\_threads++)

{

int start = clock();

int rows\_per\_thread = SIZE / num\_threads; //Деление нацело.

thread\* thr = new thread[num\_threads];

//Для всех потоков кроме последнего

int rows\_from = 1;

for (int i = 0; i < num\_threads - 1; i++)

{

thr[i] = thread(thread\_calculate, A, B, C, rows\_from, rows\_from + rows\_per\_thread - 1, SIZE);

rows\_from += (rows\_per\_thread + 1);

thr[i].join();

}

//Для посленнего потока

thr[num\_threads - 1] = thread(thread\_calculate, A, B, C, rows\_from, SIZE, SIZE);

thr[num\_threads - 1].join();

int end = clock();

ofstream file("output.txt", ios\_base::app);

file << num\_threads << " " << end - start << endl;

for (int i = 0; i < num\_threads; i++)

thr[i].~thread();

delete [] thr;

}

return 0;

}