Мордовский государственный университет имени Н.П.Огарёва

Специальность: автоматизированные системы обработки информации и управления

# ПРАКТИЧЕСКАЯ РАБОТА

### по системам с параллельной обработкой данных (С++)

**Многопоточность**

ПР – 02069964 – 090301 – 17

|  |  |
| --- | --- |
| ВЫПОЛНИЛ:  Студент 241 группы специальности АСОИУ  С. А. Константинов  \_\_\_\_\_\_\_\_\_\_\_\_ 13.07.2017 | ПРОВЕРИЛА:  ст. преподаватель кафедры АСОИУ  к.т.н. Н. П. Плотникова  \_\_\_\_\_\_\_\_\_\_\_\_ 13.07.2017 |

Саранск 2017

**Задания по разделу «Многопоточность»**

1.Реализовать приложение, выполняющее умножение элементов массива по модулю 109+7. Разбить задачу на подзадачи, выполнить распределение подзадач по потокам. Выполнить замер производительности (времени работы) программы для различного количества потоков (1-10), построить график зависимости. Объяснить полученные результаты.

|  |
| --- |
| #include <iostream>  #include <thread>  #include <chrono>  #include <vector>  #include <mutex>  #include <algorithm>  #include <ctime>  using namespace std;  mutex sum\_lock;  const int thread\_count = 5;  const int array\_size = 1e8;  const int modul = 1e9 + 7;  long long array\_mul = 1;  vector<int> my\_array(array\_size, 0);  void thread\_function(int left, int right)  {  long long local\_mul = 1;  for (int i = left; i < right; ++i)  {  if ((local\_mul \* my\_array[i]) % modul != 0) {  local\_mul = (my\_array[i] \* local\_mul) % modul;  }  local\_mul \*= my\_array[i];  }  lock\_guard<mutex> \_output\_lock(sum\_lock);  cout << "Thread #" << this\_thread::get\_id() << " is completed" << endl;  if ((array\_mul \* local\_mul) % modul != 0) {  array\_mul = (local\_mul \* array\_mul) % modul;  }  array\_mul \*= local\_mul;  }  int main()  {  srand(time(nullptr));  generate(my\_array.begin(), my\_array.end(), rand);  auto start\_time = clock();  vector<thread> ths(thread\_count);  int cur\_counter = 0, cur\_left = 0, cur\_right = 0;  int block\_size = array\_size / thread\_count;  for (auto &th : ths)  {  cur\_left = cur\_counter\*block\_size;  cur\_right = cur\_left + block\_size;  th = thread(thread\_function, cur\_left, cur\_right);  cur\_counter++;  }  if (array\_size % thread\_count != 0) {  thread th = thread(thread\_function, cur\_counter\*block\_size, array\_size);  th.join();  }  for (auto &th : ths)  th.join();    auto end\_time = clock();  cout << "Time: " << (end\_time - start\_time) / 1000.0 << endl;  return 0;  } |

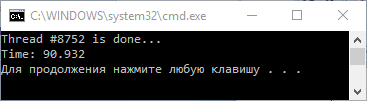


Рисунок 1 – Результат работы с 1 потоком

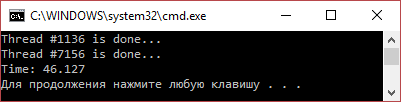


Рисунок 2 – Результат работы с 2 потоками

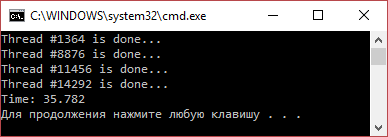


Рисунок 3 – Результат работы с 3 потоками

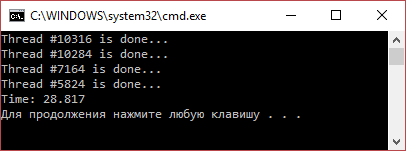


Рисунок 4 – Результат работы с 4 потоками

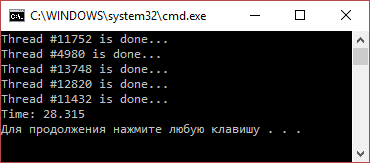


Рисунок 5 – Результат работы с 5 потоками

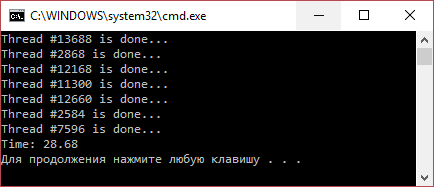


Рисунок 6 – Результат работы с 6 потоками

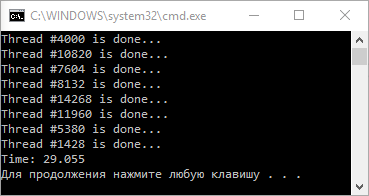


Рисунок 7 – Результат работы с 7 потоками

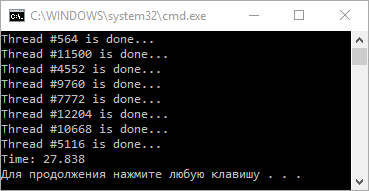


Рисунок 8 – Результат работы с 8 потоками

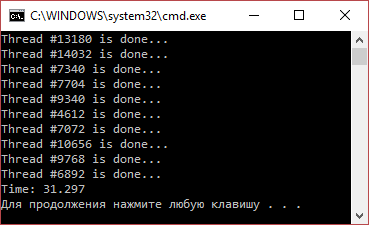


Рисунок 9 – Результат работы с 9 потоками

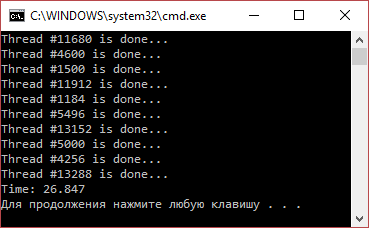


Рисунок 10 – Результат работы с 10 потоками

При увеличении вычислительных единиц время работы уменьшается.

2.Реализовать многопоточное приложение, выполняющее перемножение матриц произвольного размера. Разбить задачу на подзадачи, выполнить распределение подзадач по потокам. Выполнить замер производительности (времени работы) программы для различного количества потоков, построить график зависимости. Объяснить полученные результаты.

|  |
| --- |
| #include <iostream>  #include <thread>  #include <vector>  #include <mutex>  using namespace std;  mutex some\_mutex;  vector<vector<int>> a, b, c;  const int array\_size = 400;  const int thread\_count = 1;  void thread\_function(vector<vector<int>> m, vector<vector<int>> n, vector<vector<int>> p, int left, int right) {  lock\_guard<mutex> otput(some\_mutex);  cout << "Thread #" << this\_thread::get\_id() << " is done..." << endl;  for (int i = left ; i < right ; ++i) {  for (int j = left; j < right; ++j) {  for (int k = left; k < right; ++k) {  p[i][j] += m[i][k] \* n[k][j];  }  }  }  }  int main() {  srand(time(nullptr));  a.resize(array\_size);  b.resize(array\_size);  c.resize(array\_size);  for (int i = 0; i < array\_size; ++i) {  a[i].resize(array\_size);  b[i].resize(array\_size);  c[i].resize(array\_size);  for (int j = 0; j < array\_size; ++j) {  a[i][j] = rand();  b[i][j] = rand();  c[i][j] = 0;  }  }  vector<thread> ths(thread\_count);  int block\_size = array\_size / thread\_count;  int cur\_left = 0, cur\_right = cur\_left + block\_size;    auto start = clock();  for (auto &th : ths) {  th = thread(thread\_function, a, b, c, cur\_left, cur\_right);  cur\_left = cur\_right;  cur\_right = cur\_left + block\_size;  }  if (array\_size % thread\_count != 0) {  thread th = thread(thread\_function, a, b, c, cur\_right\*thread\_count, array\_size);  th.join();  }  for (auto &th : ths)  th.join();  auto end = clock();  cout << "Time: " << (end - start) / 1000.0 << endl;  return 0;  } |

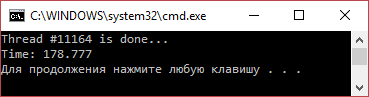
****

Рисунок 11 – Результат работы с 1 потоком.

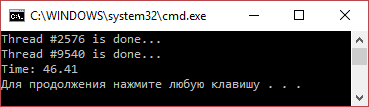
****

Рисунок 12 – Результат работы с 2 потоками.

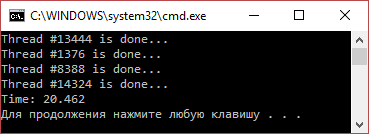
****

Рисунок 13 – Результат работы с 3 потоками.

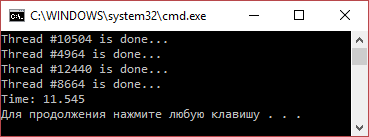
****

Рисунок 14 – Результат работы с 4 потоками.

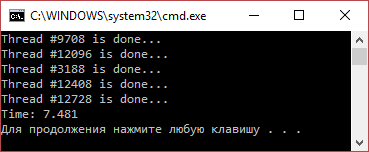
****

Рисунок 15 – Результат работы с 5 потоками.

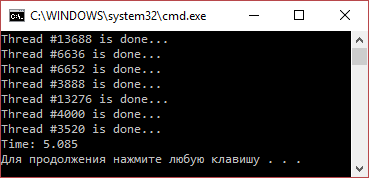
****

Рисунок 16 – Результат работы с 6 потоками.

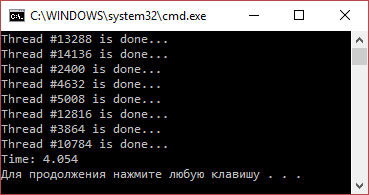


Рисунок 17 – Результат работы с 7 потоками.

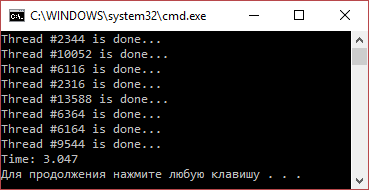


Рисунок 18 – Результат работы с 8 потоками.

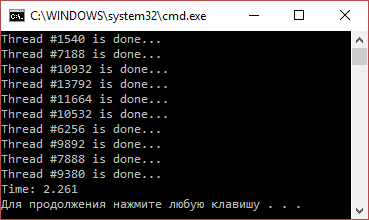


Рисунок 19 – Результат работы с 9 потоками.

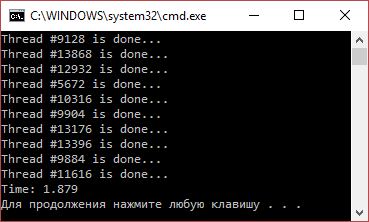


Рисунок 20 – Результат работы с 10 потоками.

При увеличении вычислительных единиц время работы уменьшается.

Процессор: Intel Pentium quad core N3540, 2.66GHz