Федеральное государственное автономное образовательное учреждение

высшего образования

«Санкт-Петербургский политехнический университет Петра Великого»

Кафедра Компьютерных систем и программных технологий

**Параллельные вычисления**

«Разработка программ с использованием pthread и MPI в языке С++»

Студент

Группа

Преподаватель

Ибаев А.Р.

13541/3

Стручков И. В.

Санкт-Петербург

2017

**Цель работы**

Научиться создавать программы с использованием многопоточных технологий. Познакомиться с работой библиотек pthread и MPI для языка C++. Проанализировать прирост производительности при использовании многопоточных библиотек.

**Постановка задачи**

Задача: Определить частоту встречи слов в тексте на русском языке.

Решить следующую задачу тремя способами:

1. Однопоточной программой.
2. Программой с использованием библиотеки pthread.
3. Программой с использованием библиотеки MPI.

**Обзор задачи**

Необходимо определить частоту встречи слов в тексте.

Для этого текст разбивается на слова, затем будет происходить подсчёт встречи слов в контейнере map.

Используемая система:

C:\Users\Bocman>systeminfo

Имя узла: MATRIX

Название ОС: Microsoft Windows 7 Максимальная

Версия ОС: 6.1.7601 Service Pack 1 сборка 7601

Изготовитель ОС: Microsoft Corporation

Параметры ОС: Изолированная рабочая станция

Сборка ОС: Multiprocessor Free

Зарегистрированный владелец: Боцман

Зарегистрированная организация:

Код продукта: 00426-OEM-8992662-00400

Дата установки: 10.07.2013, 23:11:55

Время загрузки системы: 17.06.2017, 18:11:51

Изготовитель системы: Micro-Star International Co., Ltd.

Модель системы: GE60 0NC/GE60 0ND

Тип системы: x64-based PC

Процессор(ы): Число процессоров - 1.

[01]: Intel64 Family 6 Model 58 Stepping 9 GenuineIntel ~2501 МГц

Версия BIOS: American Megatrends Inc. E16GAIMS.10E, 11.07.2012 Полный Полный объем физической памяти: 8 089 МБ

Доступная физическая память: 4 495 МБ

Виртуальная память: Макс. размер: 16 176 МБ

Виртуальная память: Доступна: 11 197 МБ

Виртуальная память: Используется: 4 979 МБ

**Однопоточная задача**

Написанная однопоточная программа представлена в листинге 1.

Листинг 1. Однопоточная программа

|  |
| --- |
| #include <iostream>  #include <vector>  #include <string>  #include <fstream>  #include <sstream>  #include <cctype>  #include <ctime>  #include <sstream>  #include <map>  using namespace std;  map <string, int> map2;  vector<string>\* filePars(string fileName)//Создание вектора слов(возможно с лишними символами).На вход получаем имя файла. На выходе вектор слов  {  vector <string>\* vec = new vector < string >; //Создаем вектор, который мы вернем  ifstream file(fileName); //Создаем входящий стрим нашей книги  if (!file.is\_open()) //если файл не открыт, тогда ретерн  return nullptr;  string word; //создаем рабочий стринг  while (file >> word) //Пока в файле есть слова, записываем слово в рабочий стринг  {  vec->push\_back(word); //в ранее созданный вектор записываем считанное слово из файла.  }  return vec; //возвращаем вектор, заполненный словами из книги  }  string formatStr(string& str)// Очищает входящий стринг от лишних символов и возвращает очищенную стрингу(очищенное слово)  {  string newStr = ""; //Рабочий стринг. В него будет записано чистое слово  for (int i = 0; i < str.size(); i++) //Цикл по символам грязногослова. От первого(0) символа до последнего(str.size())  {  if (!isalpha((unsigned char)str[i])) //Если попался символам, то ничего не далаем  continue;  char ch = tolower(str[i]); //Продолжение если попалась Буква  newStr.push\_back(ch); //Записываем букву в конец рабочего стринга  }  return newStr; //Возврящаем полученное очищенное слово  }  vector<string>\* formatFile(vector<string>\* text)//Очищает входящий (С лишними символами)вектор слов, и возвращает очищенный  {  for (string& str : \*text) //Цикл, пока в входящем векторе есть слова  {  str = formatStr(str); //Вызов функции очищения одного слова  }  return text; //Возвращаем вектор с чистыми словами  }  int сounterWords(vector <string>\* text) //Функция подсчета слов. На вход принимает вектор чистыми словами  {  if (!text->size()) //Если хотя вектор пустой, возвращаем 0  return 0;  int num = 0;  for (int i = 0; i < text->size(); i++) //Перебор слов в очищенном векторе от первого слова до последнего  {  string sT; //создаем рабочий стринг  sT = text->at(i); //в рабочий стринг записываем полученное слово из вектора в данном проходе цикла  map2[sT]++; //Записываем слово в мап  }  return num;  }  int main(int argc, char \*argv[])  {  setlocale(LC\_ALL, "rus");  cout << "Program was started" << endl;  string fileName("C:\\Users\\Bocman\\Documents\\5Kurse2SEM\\Strychkov\\1\\War2.txt");  if (argc >= 2)  {  fileName = argv[1];  }  vector<string> \*vecText = filePars(fileName); //Создаем рабочий вектор. Вызываем функцию filePars, которая разбивает книгу по словам  //и записывает их в рабочий вектор    if (vecText == nullptr)  {  cout << "File not found\n" << endl;  delete vecText;  return 1;  }  unsigned int vremPosleChtenia = clock(); //Затраченное время работы программы на чтение из файла  vecText = formatFile(vecText); //Вызываем функцию formatFile, которя очищает слова в векторе. И возвращаем  //очищенный вектор    сounterWords(vecText); //Вызываем функцию подсчёта слов из вектора слов  delete vecText; //Удаляем рабочий вектор  unsigned int vremKonca = clock(); //Записываем время работы всей программы  unsigned int vremRaboti = (vremKonca - vremPosleChtenia);//Вычисляем время работы программы без чтения файла      for (auto it = map2.begin(); it != map2.end(); ++it) { //Печатаем слова и частоту их встречи в консоль из рабочего Мапа  if (it->first != "")  cout << it->first << " : " << it->second << endl;  }  cout << "Время выполнения = " << vremKonca << endl; // время работы программы  cout << "Time = " << vremRaboti << endl; // время работы программы  return 0;  } |

**Обзор функций:**

vector<string>\* filePars(string fileName) – функция открывает файл на чтение, и разбивает содержимое файла на слова, заполняя ими возвращаемый вектор.

Входной параметр: filename - имя файла

Возвращаемое значение: вектор слов из исходного файла

string formatStr(string& str) – функция удаляет из вектора слов лишние символы и возвращает очищенное слово.

Входной параметр: str – слово вместе с символами

Возвращаемое значение: «очищенное» слово (без символов)

vector<string>\* formatFile(vector<string>\* text)- циклически вызывает функцию formatStr(), для очистки от символов каждого слова из вектора слов.

Входной параметр: text – вектор слов вместе с символами

Возвращаемое значение: текст без символов

int сounterWords(vector <string>\* text) – подсчитывает частоту встречи слов в тексте.

Входной параметр: text – текст без символов

Возвращаемое значение: число, говорящее об успешности выполнения функции

**Выполнение программы:**

Для тестирования программы использовалась отрывки из книги World of warcraft <Битва за честь>.

Отрывок 1

|  |
| --- |
| Program was started  а : 12  айлани : 14  активность : 1  алфинах : 1  ароматы : 1  …  эффект : 1  эффективное : 1  я : 25  языки : 1  языков : 1  яркий : 1  Время выполнения = 207  Время выполнения без чтения = 143  Для продолжения нажмите любую клавишу . . . |

Отрывок 2

|  |
| --- |
| Program was started  а : 14  аккуратно : 1  арбалетчиков : 1  артефактами : 1  атакуют : 1  атакующему : 1  …  это : 5  этого : 1  этот : 1  я : 10  явно : 1  ярких : 1  ясно : 1  Время выполнения = 230  Время выполнения без чтения = 151  Для продолжения нажмите любую клавишу . . . |

Из вывода программы видны все слова, встречающиеся в тексте, а также количество их появлений, время выполнения программы и время работы алгоритма без учета времени на считывание файла.

**Многопоточная программы с использованием библиотеки pthread**

Код многопоточной программы с использованием библиотеки pthread представлен в листинге 2.

Листинг 2. Многопоточная программы с использованием библиотеки pthread.

|  |
| --- |
| #include <vector>  #include <string>  #include <fstream>  #include <sstream>  #include <thread>  #include <ctime>  #include <cctype>  #include <map>  #include <iostream>  #include <windows.h>  #include <process.h>  #include <iterator>  #include <algorithm>  using namespace std;  map <string, int> map2;  HANDLE hSemaphore;  // имя семафора  const char lpSemaphoreName[] = "MySemaphore";  string formatStr(string& str)// Очищает входящий стринг от лишних символов и возвращает очищенную стрингу(очищенное слово)  {  string newStr = ""; //Рабочий стринг. В него будет записано чистое слово  for (int i = 0; i < str.size(); i++) //Цикл по символам грязногослова. От первого(0) символа до последнего(str.size())  {  if (!isalpha((unsigned char)str[i])) //Если попался символам, то ничего не далаем  continue;  char ch = tolower(str[i]); //Продолжение если попалась Буква  newStr.push\_back(ch); //Записываем букву в конец рабочего стринга  }  return newStr; //Возврящаем полученное очищенное слово  }  void formatFileV2(vector<string>\* text, int start, int end)//Очищает входящую ЧАСТЬ(С лишними символами)вектора слов, и возвращает очищенный  {  for (int i = start; i <= end; i++) //Цикл, пока в входящей Части вектора (от start слов до end слов) есть слова  {  text->at(i) = formatStr(text->at(i));//Вызов функции очищения одного слова и перезапись на грязное.  }  return;  }  vector<string>\* filePars(string fileName)//Создание вектора слов(возможно с лишними символами).На вход получаем имя файла. На выходе вектор слов  {  vector <string>\* vec = new vector < string >; //Создаем вектор, который мы вернем  ifstream file(fileName); //Создаем входящий стрим нашей книги  if (!file.is\_open()) //если файл не открыт, тогда ретерн  return nullptr;  string word; //создаем рабочий стринг  while (file >> word) //Пока в файле есть слова, записываем слово в рабочий стринг  {  vec->push\_back(word); //в ранее созданный вектор записываем считанное слово из файла.  }  return vec; //возвращаем вектор, заполненный словами из книги  }  void сounterWords2(int numth, vector <string>\* text, int start, int end)//Функция подсчета слов. На вход принимает вектор с чистыми словами.  { // И работает с участком слов от start до end  if (!text->size()) //Если хотя бы один вектор пуст  return;  if (start < 0 || end > text->size() - 1 || end < start)  return;  if (end - start + 1 < 1)  return;  int num = 0;  for (int i = start; i <= end; i++)//Перебор слов в очищенном Участке векторе от start слова до последнего(end)  {  string sT;//создаем рабочий стринг  sT = text->at(i);//в рабочий стринг записываем полученное слово из вектора в данном проходе цикла  if (WaitForSingleObject(hSemaphore, 30000) == WAIT\_FAILED) //Объявляем семафор, который не даёт обратится к участки программы  //если он занят другим потоком. Как только освободится, мы занимаем его  return;  map2[sT]++;//Записываем слово в мап  if (hSemaphore != NULL)  ReleaseSemaphore(hSemaphore, 1, NULL); //Освобождение семафора  }  return;  }  int main(int argc, char \*argv[])  {  setlocale(LC\_ALL, "rus");  std::cout << "Program was started" << endl;  string fileName("C:\\Users\\Bocman\\Documents\\5Kurse2SEM\\Strychkov\\1\\War2.txt");  int numberOfProcesses = 4;  hSemaphore = CreateSemaphore(NULL, 1, 1, lpSemaphoreName);  if (hSemaphore == NULL) {  std::cout << "Sem dont created" << endl;  return FALSE;  }  if (argc >= 3)  {  fileName = argv[1];  numberOfProcesses = atoi(argv[2]);  }  vector<string> \*vecText = filePars(fileName);//Создаем рабочий вектор. Вызываем функцию filePars, которая разбивает книгу по словам  //и записывает их в рабочий вектор  if (vecText == nullptr)  {  std::cout << "error in reading data" << endl;  return 1;  }  unsigned int vremPosleChtenia = clock(); //Затраченное время работы программы на чтение из файла  int textSize = vecText->size(); //Вызываем функцию formatFile, которя очищает слова в векторе. И возвращаем  //очищенный вектор  if (vecText->size() < numberOfProcesses)  {  //Не делим на процессы  сounterWords2(1, vecText, 0, textSize - 1); //Числитель - число появлений конкретной nGramm-ы  }  else  {  //Делим на процессы  vector<thread\*> vecThreads; //Создаем вектор потоков  int step = textSize / numberOfProcesses; //Пример: 8-длина текста, 2- число проц. 8/2=4  int start = 0; //от 0  int end = step - 1; //до 3 и от 4 до 7  for (int i = 1; i <= numberOfProcesses; i++) //Цикл создание Потоков  {  auto th = new std::thread(formatFileV2, vecText, start, end); //Создаем поток с функцией formatFileV2 и передаем границы работы  vecThreads.push\_back(th); //Запись Потока в вектор потоков  start = end;  //последний процесс захватывает все слова до конца  if (i == numberOfProcesses)  end = textSize - 1;  else  end = end + step;  }  for (auto &th : vecThreads) //Перебираем вектор Потоков  {  th->join(); //Ожидаем завершение потока  delete th;  }  vecThreads.clear();  step = textSize / numberOfProcesses; //Пример: 8-длина текста, 2- число проц. 8/2=4  start = 0; //от 0  end = step - 1; //до 3 и от 4 до 7  for (int i = 1; i <= numberOfProcesses; i++)  {  //if(i==1)  auto th = new std::thread(сounterWords2, i, vecText, start, end);  vecThreads.push\_back(th);  start = end;  //последний процесс захватывает все слова доконце  if (i == numberOfProcesses)  end = textSize - 1;  else  end = end + step;  }  for (auto &th : vecThreads)  {  th->join();  delete th;  }  }  delete vecText;  unsigned int vremKonca = clock();  unsigned int vremRaboti = (vremKonca - vremPosleChtenia);//в секундах  for (auto it = map2.begin(); it != map2.end(); ++it) {  //if(it->first!="")  cout << it->first << " : " << it->second << endl;  }  cout << "Время работы программы = " << vremKonca << endl; // время работы программы  cout << "Time = " << vremRaboti << endl; // время работы программы без чтения  CloseHandle(hSemaphore);  system("pause");  ofstream fout("Statistics.txt", ios\_base::app); // открываем файл для добавления информации к концу файла  if (!fout.is\_open()) // если файл не открыт  cout << "Файл не может быть открыт!\n"; // сообщить об этом  fout << vremRaboti << endl;  fout.close();  return 0;  } |

Программа по умолчанию использует 4 потока.

Для реализации многопоточности используется класс thread.

Класс thread предоставляет один поток выполнения. Потоки позволяют нескольким фрагментам кода работать асинхронно и одновременно.

Точки распараллеливания:

1. Функция formatFile запускается в n потоках. В каждом потоке считывается свой промежуток слов в исходном векторе.
2. Функция сounterWords запускается в n потоках. В каждом потоке подсчитывается свой промежуток слов в исходном векторе.

Функции программы подобны функциям однопоточного приложения и выполняют те же задачи.

Функции formatFile и сounterWords изменены для того чтобы их можно было использовать при распараллеливании.

void formatFileV2(vector<string>\* text, int start, int end)

Добавленные входные параметры:

start– начало проверки слов

end – конец проверки слов

В программу добавлено принятия третьего аргумента – число потоков.

Для тестирования программы использовалась отрывки из книги World of warcraft <Битва за честь>.

Отрывок 1

|  |
| --- |
| Program was started  а : 12  айлани : 14  активность : 1  алфинах : 1  ароматы : 1  …  эффект : 1  эффективное : 1  я : 25  языки : 1  языков : 1  яркий : 1  Время работы программы = 186  Время работы программы без чтения = 122  Для продолжения нажмите любую клавишу . . . |

Отрывок 2

|  |
| --- |
| Program was started  а : 14  аккуратно : 1  арбалетчиков : 1  артефактами : 1  …  это : 5  этого : 1  этот : 1  я : 10  явно : 1  ярких : 1  ясно : 1  Время работы программы = 204  Время работы программы без чтения = 132  Для продолжения нажмите любую клавишу . . . |

Как видно из результата, выполнение многопоточной программы оказалось быстрее её однопоточного аналога:

1-132/151 = 0.12

Таким образом, многопоточная программа быстрее однопоточной на 12%.

**Многопоточная программа с использованием библиотеки MPI**

Код многопоточной программы с использованием библиотеки MPI представлен в листинге 3.

Листинг 3. Многопоточная программа с использованием библиотеки MPI.

|  |
| --- |
| #include <mpi.h>  #include <fstream>  #include <string>  #include <iostream>  #include <vector>  #include <vector>  #include <algorithm>  #include <iterator>  #include <iomanip>  #include <vector>  #include <string>  #include <fstream>  #include <sstream>  #include <thread>  #include <ctime>  #include <cctype>  #include <map>  #include <iostream>  #include <windows.h>  #include <process.h>  #include <iterator>  #include <algorithm>  using namespace std;  map <string, int> map2;  HANDLE hSemaphore;  // имя семафора  const char lpSemaphoreName[] = "MySemaphore";  string formatStr(string& str)// Очищает входящий стринг от лишних символов и возвращает очищенную стрингу(очищенное слово)  {  string newStr = ""; //Рабочий стринг. В него будет записано чистое слово  for (int i = 0; i < str.size(); i++) //Цикл по символам грязногослова. От первого(0) символа до последнего(str.size())  {  if (!isalpha((unsigned char)str[i])) //Если попался символам, то ничего не далаем  continue;  char ch = tolower(str[i]); //Продолжение если попалась Буква  newStr.push\_back(ch); //Записываем букву в конец рабочего стринга  }  return newStr; //Возврящаем полученное очищенное слово  }  void formatFileV2(vector<string>\* text, int start, int end)//Очищает входящую ЧАСТЬ(С лишними символами)вектора слов, и возвращает очищенный  {  for (int i = start; i <= end; i++) //Цикл, пока в входящей Части вектора (от start слов до end слов) есть слова  {  text->at(i) = formatStr(text->at(i));//Вызов функции очищения одного слова и перезапись на грязное.  }  return;  }  vector<string>\* filePars(string fileName)//Создание вектора слов(возможно с лишними символами).На вход получаем имя файла. На выходе вектор слов  {  vector <string>\* vec = new vector < string >; //Создаем вектор, который мы вернем  ifstream file(fileName); //Создаем входящий стрим нашей книги  if (!file.is\_open()) //если файл не открыт, тогда ретерн  return nullptr;  string word; //создаем рабочий стринг  while (file >> word) //Пока в файле есть слова, записываем слово в рабочий стринг  {  vec->push\_back(word); //в ранее созданный вектор записываем считанное слово из файла.  }  return vec; //возвращаем вектор, заполненный словами из книги  }  void сounterWords2(int numth, vector <string>\* text, int start, int end)//Функция подсчета слов. На вход принимает вектор с чистыми словами.  { // И работает с участком слов от start до end. Лова отправляются главному потоку для подсчёта  if (!text->size()) //Если хотя бы один вектор пуст  return;  if (start < 0 || end > text->size() - 1 || end < start)  return;  if (end - start + 1 < 1)  return;  int num = 0;  for (int i = start; i <= end; i++)//Перебор слов в очищенном Участке векторе от start слова до последнего(end)  {  string sT;//создаем рабочий стринг  sT = text->at(i);//в рабочий стринг записываем полученное слово из вектора в данном проходе цикла  char buf[10000];  //double rexM = 2;  //sprintf(buf, "%i", start);  strcpy(buf, sT.c\_str());  MPI\_Send(buf, sizeof(buf), MPI\_CHAR, 0, 0, MPI\_COMM\_WORLD); //Функция отправки MPI. buf содержит отправляемое слово, 0 - адрес главного процесса.  }    //cout << buf << endl;  return;  }  int main(int argc, char \*\*argv) {  setlocale(LC\_ALL, "rus");  std::cout << "Program was started" << endl;  string fileName("C:\\Users\\Bocman\\Documents\\5Kurse2SEM\\Strychkov\\1\\War2.txt");  int numberOfProcesses = 4;  if (argc >= 3)  {  fileName = argv[1];  numberOfProcesses = atoi(argv[2]);  }  vector<string> \*vecText = filePars(fileName); //Создаем рабочий вектор. Вызываем функцию filePars, которая разбивает книгу по словам  //и записывает их в рабочий вектор  if (vecText == nullptr)  {  std::cout << "error in reading data" << endl;  return 1;  }  unsigned int timeAfterReading = clock(); //Затраченное время работы программы на чтение из файла  int textSize = vecText->size(); //Вызываем функцию formatFile, которя очищает слова в векторе. И возвращаем  //очищенный вектор  MPI\_Init(&argc, &argv); //Иницианилизируем MPI  int rank, size;  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank); //Получаем id текущего процессора  MPI\_Comm\_size(MPI\_COMM\_WORLD, &size); //Получаем колличество запущенных процессоров  numberOfProcesses = size-1;  if (rank) //Условие. Если это не главный процессор.  {  int step = textSize / numberOfProcesses; //Пример: 8-длина текста, 2- число проц. 8/2=4  int start = 0; //от 0  int end = step - 1; //до 3 и от 4 до 7  //cout <<"Text size: "<< textSize << endl;  for (int i = 1; i <= numberOfProcesses; i++)  {  if (i == rank){ //Проверка на соответствие участка книги текущему процессу  formatFileV2(vecText, start, end);  cout << "I Process " << rank << " Start:" << start << " End:" << end << endl;  }  start = end;  //последний процесс захватывает все слова до конца  if (i == numberOfProcesses)  end = textSize - 1;  else  end = end + step;  }  step = textSize / numberOfProcesses; //Пример: 8-длина текста, 2- число проц. 8/2=4  start = 0; //от 0  end = step - 1; //до 3 и от 4 до 7  for (int i = 1; i <= numberOfProcesses; i++)  {  if (i == rank)//Проверка на соответствие участка книги текущему процессу  сounterWords2( i, vecText, start, end);  start = end;  //последний процесс захватывает все слова доконце  if (i == numberOfProcesses)  end = textSize - 1;  else  end = end + step;  }    }  else { //Запуск Главного Процессора    cout << "Process 0 started" << endl;  int step = textSize / numberOfProcesses; //Пример: 8-длина текста, 2- число проц. 8/2=4  int start = 0; //от 0  int end = step - 1; //до 3 и от 4 до 7  //cout << "Text size: " << textSize << endl;  for (int i = 1; i <= numberOfProcesses; i++) //Узнаем сколько всего будет принято слов  {    start = end;  //последний процесс захватывает все слова до конца  if (i == numberOfProcesses)  end = textSize - 1;  else  end = end + step;  }  for (int i(1); i < end; ++i) //Цикл принятия слов от всех не нлавных Процессов  {  MPI\_Status s; // Рабочая переменная MPI процесса  MPI\_Probe(MPI\_ANY\_SOURCE, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &s); // Узнаем тип принимаемого объекта и записываем его в Рабочую переменную MPI процесса s  int count;  MPI\_Get\_count(&s, MPI\_CHAR, &count); // Узнаем длину принимаего соощения    char \*buf = new char[count];  MPI\_Recv(buf, count, MPI\_CHAR, // Принимаем сообщение(слово)  MPI\_ANY\_SOURCE, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &s);  //cout << "Process " << s.MPI\_SOURCE << ": "<< "End work" << endl;  //double x = atof(buf);  string str(buf); // Конвертируем полученный char в string  //cout << "Info "<<buf <<endl;  map2[str]++; // Записываем в мап для подсчета  delete[] buf;    }  unsigned int timeEnd = clock();  ofstream fout("Statistics.txt", ios\_base::app); // открываем файл для добавления информации к концу файла  if (!fout.is\_open()) // если файл не открыт  cout << "Файл не может быть открыт!\n";  for (auto it = map2.begin(); it != map2.end(); ++it) {  //if(it->first!="")  //cout << it->first << " : " << it->second << endl;/////////////////////////////////////////////////////////////////////////////////////    fout << it->first << " : " << it->second << endl;    }  fout.close();  unsigned int workTime = (timeEnd - timeAfterReading);// / 1000; //в секундах  cout << "Time = " << workTime << endl; // время работы программы  }  MPI\_Finalize(); // Завершение MPI        return 0;  } |

Программа построена на основе многопоточной программы и имеет большие изменения по сравнению с ней. Запуск библиотеки MPI происходит функцией - MPI\_Init(), завершение - MPI\_Finalize(). Точку распараллеливания было решено оставить. Каждый процесс MPI считывает текст из файла, удаляет лишние символы и отправляет чистое слово на подсчёт главному процессу.

Так же пропала необходимость в семафоре, так как к ресурсу map не происходит одновременного обращения нескольких процессов.

Для тестирования программы использовалась отрывки из книги World of warcraft <Битва за честь>.

Отрывок 1

|  |
| --- |
| C:\Users\Bocman\Documents\5Kurse2SEM\Strychkov\A\MPI2\Debug>mpiexec -n 5 MPI2.exe  Program was started  Program was started  Program was started  Program was started  Program was started  Process 0 started  I Process 2 Start:553 End:1107  I Process 3 Start:1107 End:1661  I Process 4 Start:1661 End:2215  I Process 1 Start:0 End:553 |

Отрывок 2

|  |
| --- |
| C:\Users\Bocman\Documents\5Kurse2SEM\Strychkov\A\MPI2\Debug>mpiexec -n 5 MPI2.exe  Program was started  Program was started  Program was started  Program was started  Program was started  Process 0 started  I Process 3 Start:1151 End:1727  I Process 2 Start:575 End:1151  I Process 1 Start:0 End:575  I Process 4 Start:1727 End:2303  Time = 119 Для продолжения нажмите любую клавишу . . . |

Как и в случае с предыдущей программой разница в скорости выполнения оказалась заметна

1 - 119/132 = 0.10

Прирост производительности 10%.

**Многократный запуск программ и подсчет вероятностных характеристик**

Для подсчета вероятностных характеристик был создан скрипт, запускающий программу 100 раз и подсчитывающий временя ее исполнения. На основе полученных данных подсчитывается математическое ожидание, дисперсия и доверительных интервал.

Скрипт представлен в листинге 3

Листинг 3. Скрипт многократного запуска программы.

|  |
| --- |
| PIPE = subprocess.PIPE  for threads in [1,2,4,8]:  timeList = []  for num in range(numRepeats):  p = subprocess.Popen([programm, fileName ,str(threads)], stdout=PIPE)  for line in p.stdout:  if 'Time = ' in line :  timeList.append(int(line.split()[-1]))  m=sum(timeList)/numRepeats  disp = 0.00  for val in timeList :  disp = disp + (val - m) \*\* 2  if numRepeats == 1:  disp = disp / numRepeats  else :  disp = disp / ( numRepeats - 1)  sigma = sqrt(disp)  t=1.984  interHigh = m + t\*(sigma/(sqrt(numRepeats)))  interLow = m - t\*(sigma/(sqrt(numRepeats))) |

На основе выводов данного скрипта была построена сводная таблица результатов для всех программ:

Табл.1. Сводная таблица результатов для 100 запусков каждой программы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Число потоков | Мат. Ожид. | Дисперси | Дов. Интер. 0.95% |
| Simple | 1 | 151 | 11 | [150,3 – 151,6] |
| pThread | 1 | 156 | 24,45 | [155 – 156,9] |
|  | 2 | 135 | 24,61 | [134 – 135,9] |
|  | 4 | **130** | 37,61 | [128,7 – 131,2] |
|  | 8 | 132 | 14,06 | [131,2 – 132,7] |
| MPI | 1 | 147 | 550,76 | [142,65 – 151,92] |
|  | 2 | 128 | 351,82 | [124,91 – 132,36] |
|  | 4 | **124** | 419,00 | [120,67 – 128,80] |
|  | 8 | 159 | 1259,14 | [152,33 – 166,42] |

Из таблицы 1 видно, что многопоточные приложения выигрывают по скорости выполнения у однопоточного. Так же видно, что программа с библиотекой MPI выигрывает у программы с использованием библиотеки pThread. Увеличение числа потоков до 8 не даёт результата, т.к. процессор используемой системы имеет 4 логических потока.

Данные результаты так же, зависят от загруженности системы в конкретный момент времени, что может серьезно влиять на скорость выполнения программ.

**Вывод**

В ходе работы создано три программы на языке С++ для решения задачи определения частоты встречи слов в тексте. В ходе работы изучены и использованы библиотеки многопоточного программирования pThread и MPI. Реализованные на их основе решения, могут легко изменять количество исполняющих потоков.

Полученные решения протестированы на большом входном файле. Для одного набора входных данных проведен анализ вероятностных характеристик (табл.1). В результате экспериментов установлено, использование библиотеки MPI намного сложнее, чем pThread, а прирост производительности составил менее 10%.

Использование библиотеки MPI позволяет распараллелить программу на несколько компьютеров, но в рамках данного эксперимента нет такой необходимости, вследствие чего результаты увеличения производительности не так велики.

Так как приложение предполагает запись в одну и ту же область данных, были использованы средства синхронизации потоков.