Санкт-Петербургский Политехнический университет

Институт Компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

**Отчет по курсовому проекту**

**Дисциплина**: Параллельные вычисления

**Тема**: Вычисление пересечения трех множеств.

Выполнил студент гр. 13541/4 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. И. Шайтан

(подпись)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_ И. В. Стручков

(подпись)

“\_\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2017 г.

Санкт-Петербург

2017

**Выполнение работы:**

<http://kspt.icc.spbstu.ru/course/parallel>

**Постановка задачи**

**Задача:** вычислить пересечение трех множеств.

Решить следующую задачу тремя способами.

**Разработка алгоритма**

У нас имеется три множества. Для начала мы каждое множество отсортируем. Если множества не отсортированы, мы может их отсортировать. После этого берем каждый элемент первого множества и смотрим есть ли данный элемент в двух других множествах.

**Используемое оборудование**

Linux Mint 18. Среда Qt Creator (Community)

Процессор Intel-Core-i5-3210M - двухъядерный процессор (4 потока)

**Однопоточная задача**

**Создание последовательной программы, реализующей алгоритм**

Пример возможной реализации приведен ниже.

|  |
| --- |
| #include <QCoreApplication>  #include <QTime>  #include <QDebug>  #include <iostream>  #include <cstdio>  using namespace std;  const int NUM\_REPEAT = 10;  const int SET\_SEED = 1;  const int MIN\_ELEMENT = 1;  const int MAX\_ELEMENT = 500000;  const int SET\_NAME\_LEN = 16;  struct set\_t  {  char name[SET\_NAME\_LEN];  int size;  int element[MAX\_ELEMENT];  }  ;  set\_t set[4];  set\_t \*A = &set[0];  set\_t \*B = &set[1];  set\_t \*C = &set[2];  set\_t \*S = &set[3];  void set\_create(set\_t \*set, char \*name, int nummod)  {  int e;  strncpy(set->name, name, SET\_NAME\_LEN);  set->size = 0;  for ( e = MIN\_ELEMENT; e < MAX\_ELEMENT; e++ )  {  if (e % nummod == 0)  {  set->element[set->size] = e;  set->size++;  }  }  }  int set\_save(set\_t \*set)  {  FILE \*pf;  int i;  char filename[256];  snprintf(filename, 256, "simple\_%s.txt", set->name);  if ( (pf = fopen(filename, "w")) == NULL )  {  perror("fopen()");  return -1;  }  fprintf(pf, "size = %d\n", set->size);  for ( i = 0; i < set->size; i++ )  {  fprintf(pf, "%d\n", set->element[i]);  }  fclose(pf);  return 0;  }  int set\_is\_belonged(set\_t \*set, int e)  {  int i;  for ( i = 0; i < set->size; i++ )  {  if ( set->element[i] == e )  return 1;  }  return 0;  }  void set\_intersection3(set\_t \*result, char \*name, set\_t \*set1, set\_t \*set2, set\_t \*set3)  {  int i;  int e;  strncpy(result->name, name, SET\_NAME\_LEN);  result->size = 0;  for ( i = 0; i < set1->size; i++ )  {  e = set1->element[i];  if ( set\_is\_belonged(set2, e) && set\_is\_belonged(set3, e) )  {  result->element[result->size] = e;  result->size++;  }  }  }  int main(int argc, char \*argv[])  {  set\_create(A, "a", 10);  set\_create(B, "b", 25);  set\_create(C, "c", 4);  set\_save(A);  set\_save(B);  set\_save(C);  int timeList[NUM\_REPEAT];  int timeListSum = 0;  int mo = 0;  int disp = 0;  for (int i = 0; i < 10; i++) {  QTime time;  time.start () ;  set\_intersection3(S, "s", A, B, C);  timeList[i] = time.elapsed();  timeListSum += time.elapsed();  qDebug() << "Время работы функции set\_intersection3(...) равно "  << time.elapsed()  << "миллисекунд"  << endl;  }  mo = timeListSum / NUM\_REPEAT;  for(int i = 0; i < NUM\_REPEAT; i++) {  disp = disp + pow((timeList[i] - mo),2);  }  disp = disp / (NUM\_REPEAT - 1);  int sigma = sqrt(disp);  float t = 2.2281;  int interHigh = mo + t \* (sigma/(sqrt(NUM\_REPEAT)));  int interLow = mo - t \* (sigma/(sqrt(NUM\_REPEAT)));  qDebug() << "Математическое ожидание работы функции set\_intersection3(...) равно "  << mo  << endl;  qDebug() << "Дисперсия работы функции функции set\_intersection3(...) равно "  << disp  << endl;  qDebug() << "Доверительный интервал set\_intersection3(...) равно "  << interLow  << " - "  << interHigh  << endl;  set\_save(S);  return EXIT\_SUCCESS;  } |

Обзор функций:

**Выполнение программы:**

|  |
| --- |
| Время работы функции set\_intersection3(...) равно 3764 миллисекунд  Время работы функции set\_intersection3(...) равно 3935 миллисекунд  Время работы функции set\_intersection3(...) равно 3785 миллисекунд  Время работы функции set\_intersection3(...) равно 3876 миллисекунд  Время работы функции set\_intersection3(...) равно 3848 миллисекунд  Время работы функции set\_intersection3(...) равно 3798 миллисекунд  Время работы функции set\_intersection3(...) равно 3723 миллисекунд  Время работы функции set\_intersection3(...) равно 3739 миллисекунд  Время работы функции set\_intersection3(...) равно 3752 миллисекунд  Время работы функции set\_intersection3(...) равно 3748 миллисекунд  Математическое ожидание работы функции set\_intersection3(...) равно 3796  Дисперсия работы функции функции set\_intersection3(...) равно 4710  Доверительный интервал set\_intersection3(...) равно 3748 - 3843 |

**Многопоточная программа с использованием библиотеки pthread**

Добавляем в .pro файл проекта.

QMAKE\_CXXFLAGS += -std=c++0x -pthread

LIBS += -pthread

Пример возможной реализации приведен ниже

|  |
| --- |
| #include <QCoreApplication>  #include <QTime>  #include <QDebug>  #include <iostream>  using namespace std;  const int NUM\_REPEAT = 10;  const int SET\_SEED = 1;  const int MIN\_ELEMENT = 1;  const int MAX\_ELEMENT = 500000;  const int SET\_NAME\_LEN = 16;  struct set\_t  {  char name[SET\_NAME\_LEN];  int size;  int element[MAX\_ELEMENT];  };  set\_t set[4];  set\_t \*A = &set[0];  set\_t \*B = &set[1];  set\_t \*C = &set[2];  set\_t \*S = &set[3];  const int MIN\_THREADS = 1;  const int MAX\_THREADS = 8;  struct thread\_data\_t  {  pthread\_mutex\_t iterator;  pthread\_mutex\_t store;  int i;  set\_t \*set1;  set\_t \*set2;  set\_t \*set3;  set\_t \*result;  };  int num\_threads = MIN\_THREADS;  void set\_create(set\_t \*set, char \*name, int nummod)  {  int e;  strncpy(set->name, name, SET\_NAME\_LEN);  set->size = 0;  for ( e = MIN\_ELEMENT; e < MAX\_ELEMENT; e++ )  {  if (e % nummod == 0)  {  set->element[set->size] = e;  set->size++;  }  }  }  int set\_save(set\_t \*set)  {  FILE \*pf;  int i;  char filename[256];  snprintf(filename, 256, "simple\_%s.txt", set->name);  if ( (pf = fopen(filename, "w")) == NULL )  {  perror("fopen()");  return -1;  }  fprintf(pf, "size = %d\n", set->size);  for ( i = 0; i < set->size; i++ )  {  fprintf(pf, "%d\n", set->element[i]);  }  fclose(pf);  return 0;  }  int set\_is\_belonged(set\_t \*set, int e)  {  int i;  for ( i = 0; i < set->size; i++ )  {  if ( set->element[i] == e )  return 1;  }  return 0;  }  static int next\_i(thread\_data\_t \*data)  {  int i;  pthread\_mutex\_lock(&(data->iterator));  i = data->i;  data->i++;  pthread\_mutex\_unlock(&(data->iterator));  return i;  }  static void \*thread\_intersect3(void \*ptr)  {  int i, k;  int e;  thread\_data\_t \*data = (thread\_data\_t \*)ptr;  for ( i = next\_i(data); i < data->set1->size; i = next\_i(data) )  {  e = data->set1->element[i];  if ( set\_is\_belonged(data->set2, e) && set\_is\_belonged(data->set3, e) )  {  pthread\_mutex\_lock(&(data->store));  data->result->element[data->result->size] = e;  data->result->size++;  pthread\_mutex\_unlock(&(data->store));  }  }  return NULL;  }  void set\_intersection3(set\_t \*result, char \*name, set\_t \*set1, set\_t \*set2, set\_t \*set3)  {  int i;  pthread\_t thread[MAX\_THREADS];  thread\_data\_t data;  strncpy(result->name, name, SET\_NAME\_LEN);  result->size = 0;  pthread\_mutex\_init(&(data.iterator), NULL);  pthread\_mutex\_init(&(data.store), NULL);  data.i = 0;  data.set1 = set1;  data.set2 = set2;  data.set3 = set3;  data.result = result;  for ( i = 1; i < num\_threads; i++ )  {  if ( pthread\_create(&thread[i], NULL, thread\_intersect3, (void \*)&data) < 0 )  {  perror("pthread\_create()");  exit(EXIT\_FAILURE);  }  }  thread\_intersect3((void \*)&data);  for ( i = 1; i < num\_threads; i++ )  {  pthread\_join(thread[i], NULL);  }  }  int main(int argc, char \*argv[])  {  if ( argc > 1 )  {  num\_threads = atoi(argv[1]);  if ( (num\_threads < MIN\_THREADS) || (num\_threads > MAX\_THREADS) )  {  num\_threads = MIN\_THREADS;  }  }  set\_create(A, "a", 10);  set\_create(B, "b", 25);  set\_create(C, "c", 4);  set\_save(A);  set\_save(B);  set\_save(C);  int timeList[NUM\_REPEAT];  int timeListSum = 0;  int mo = 0;  int disp = 0;  for (int i = 0; i < 10; i++) {  QTime time;  time.start () ;  set\_intersection3(S, "s", A, B, C);  timeList[i] = time.elapsed();  timeListSum += time.elapsed();  qDebug() << "Время работы функции set\_intersection3(...) равно "  << time.elapsed()  << "миллисекунд"  << endl;  }  mo = timeListSum / NUM\_REPEAT;  for(int i = 0; i < NUM\_REPEAT; i++) {  disp = disp + pow((timeList[i] - mo),2);  }  disp = disp / (NUM\_REPEAT - 1);  int sigma = sqrt(disp);  float t = 2.2281;  int interHigh = mo + t \* (sigma/(sqrt(NUM\_REPEAT)));  int interLow = mo - t \* (sigma/(sqrt(NUM\_REPEAT)));  qDebug() << "Математическое ожидание работы функции set\_intersection3(...) равно "  << mo  << endl;  qDebug() << "Дисперсия работы функции функции set\_intersection3(...) равно "  << disp  << endl;  qDebug() << "Доверительный интервал set\_intersection3(...) равно "  << interLow  << " - "  << interHigh  << endl;  set\_save(S);  return EXIT\_SUCCESS;  } |

Обзор функций

int pthread\_mutex\_init(pthread\_mutex\_t \*mutex, const pthread\_mutexattr\_t \*attr);

где первый аргумент – указатель на мьютекс, а второй – атрибуты мьютекса. Если указан NULL, то используются атрибуты по умолчанию.

Новый поток создаётся с помощью функции pthread\_create

|  |
| --- |
| int pthread\_create(\*ptherad\_t, const pthread\_attr\_t \*attr, void\* (\*start\_routine)(void\*), void \*arg); |

Функция получает в качестве аргументов указатель на поток, переменную типа pthread\_t, в которую, в случае удачного завершения сохраняет id потока. pthread\_attr\_t – атрибуты потока. В случае если используются атрибуты по умолчанию, то можно передавать NULL. start\_routin – это непосредственно та функция, которая будет выполняться в новом потоке. arg – это аргументы, которые будут переданы функции.

Поток может выполнять много разных дел и получать разные аргументы. Для этого функция, которая будет запущена в новом потоке, принимает аргумент типа void\*. За счёт этого можно обернуть все передаваемые аргументы в структуру. Возвращать значение можно также через передаваемый аргумент.

Функция

int pthread\_join(pthread\_t thread, void \*\*value\_ptr);

Откладывает выполнение вызывающего (эту функцию) потока, до тех пор, пока не будет выполнен поток thread. Когда pthread\_join выполнилась успешно, то она возвращает 0. Если поток явно вернул значение (это то самое значение SUCCESS, из нашей функции), то оно будет помещено в переменную value\_ptr.

После создания мьютекса он может быть захвачен с помощью функции

int pthread\_mutex\_lock(pthread\_mutex\_t \*mutex);

После этого участок кода становится недоступным остальным потокам – их выполнение блокируется до тех пор, пока мьютекс не будет освобожден. Освобождение должен провести поток, заблокировавший мьютекс, вызовом

int pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex);

**Выполнение программы:**

|  |
| --- |
| Время работы функции set\_intersection3(...) равно 3871 миллисекунд  Время работы функции set\_intersection3(...) равно 3690 миллисекунд  Время работы функции set\_intersection3(...) равно 3720 миллисекунд  Время работы функции set\_intersection3(...) равно 3716 миллисекунд  Время работы функции set\_intersection3(...) равно 3705 миллисекунд  Время работы функции set\_intersection3(...) равно 3692 миллисекунд  Время работы функции set\_intersection3(...) равно 3702 миллисекунд  Время работы функции set\_intersection3(...) равно 3826 миллисекунд  Время работы функции set\_intersection3(...) равно 3892 миллисекунд  Время работы функции set\_intersection3(...) равно 3900 миллисекунд  Математическое ожидание работы функции set\_intersection3(...) равно 3771  Дисперсия работы функции функции set\_intersection3(...) равно 7956  Доверительный интервал set\_intersection3(...) равно 3708 - 3833 |

**Многопоточная программа с использованием библиотеки OpenMP**

Добавляем в .pro файл.

QMAKE\_CXXFLAGS += -fopenmp

LIBS += -fopenmp

Пример возможной реализации приведен ниже

|  |
| --- |
| #include <QCoreApplication>  #include <QTime>  #include <QDebug>  #include <iostream>  #include <omp.h>  #include <algorithm>  using namespace std;  const int NUM\_REPEAT = 50;  const int SET\_SEED = 1;  const int MIN\_ELEMENT = 1;  const int MAX\_ELEMENT = 500000;  const int SET\_NAME\_LEN = 16;  typedef struct  {  char name[SET\_NAME\_LEN];  int size;  int element[MAX\_ELEMENT];  }  set\_t;  set\_t set[4];  set\_t \*A = &set[0];  set\_t \*B = &set[1];  set\_t \*C = &set[2];  set\_t \*S = &set[3];  void set\_create(set\_t \*set, char \*name, int nummod)  {  int e;  strncpy(set->name, name, SET\_NAME\_LEN);  set->size = 0;  for ( e = MIN\_ELEMENT; e < MAX\_ELEMENT; e++ )  {  if (e % nummod == 0)  {  set->element[set->size] = e;  set->size++;  }  }  }  int set\_save(set\_t \*set)  {  FILE \*pf;  int i;  char filename[256];  snprintf(filename, 256, "simple\_%s.txt", set->name);  if ( (pf = fopen(filename, "w")) == NULL )  {  perror("fopen()");  return -1;  }  fprintf(pf, "size = %d\n", set->size);  for ( i = 0; i < set->size; i++ )  {  fprintf(pf, "%d\n", set->element[i]);  }  fclose(pf);  return 0;  }  int set\_is\_belonged(set\_t \*set, int e)  {  int i;  for ( i = 0; i < set->size; i++ )  {  if ( set->element[i] == e )  return 1;  }  return 0;  }  void set\_intersection3(set\_t \*result, char \*name, set\_t \*set1, set\_t \*set2, set\_t \*set3)  {  int i;  int e;  strncpy(result->name, name, SET\_NAME\_LEN);  result->size = 0;  omp\_set\_num\_threads(4);  #pragma omp parallel for private(e)  for ( i = 0; i < set1->size; i++ )  {  e = set1->element[i];  if ( set\_is\_belonged(set2, e) && set\_is\_belonged(set3, e) )  {  {  result->element[result->size] = e;  result->size++;  }  }  }  }  int main(int argc, char \*argv[])  {  set\_create(A, "a", 10);  set\_create(B, "b", 25);  set\_create(C, "c", 4);  set\_save(A);  set\_save(B);  set\_save(C);  int timeList[NUM\_REPEAT];  int timeListSum = 0;  int mo = 0;  int disp = 0;  for (int i = 0; i < NUM\_REPEAT; i++) {  QTime time;  time.start () ;  set\_intersection3(S, "s", A, B, C);  timeList[i] = time.elapsed();  timeListSum += time.elapsed();  qDebug() << "Время работы функции set\_intersection3(...) равно "  << time.elapsed()  << "миллисекунд"  << endl;  }  mo = timeListSum / NUM\_REPEAT;  for(int i = 0; i < NUM\_REPEAT; i++) {  disp = disp + pow((timeList[i] - mo),2);  }  disp = disp / (NUM\_REPEAT - 1);  int sigma = sqrt(disp);  //float t = 2.2281;  float t = 2.0086;  int interHigh = mo + t \* (sigma/(sqrt(NUM\_REPEAT)));  int interLow = mo - t \* (sigma/(sqrt(NUM\_REPEAT)));  qDebug() << "Математическое ожидание работы функции set\_intersection3(...) равно "  << mo  << endl;  qDebug() << "Дисперсия работы функции функции set\_intersection3(...) равно "  << disp  << endl;  qDebug() << "Доверительный интервал set\_intersection3(...) равно "  << interLow  << " - "  << interHigh  << endl;  sort(S->element, S->element+S->size);  set\_save(S);  return EXIT\_SUCCESS;  } |

Программа построена на основе однопоточной программы. Имеет незначительные изменения. Добавлены директивы компилятора #pragma.

Кроме этого запуск программы производился с ключом -fopenmp.

**Выполнение программы:**

|  |
| --- |
| Время работы функции set\_intersection3(...) равно 2210 миллисекунд  Время работы функции set\_intersection3(...) равно 2244 миллисекунд  Время работы функции set\_intersection3(...) равно 2254 миллисекунд  Время работы функции set\_intersection3(...) равно 2160 миллисекунд  Время работы функции set\_intersection3(...) равно 2234 миллисекунд  Время работы функции set\_intersection3(...) равно 2191 миллисекунд  Время работы функции set\_intersection3(...) равно 2185 миллисекунд  Время работы функции set\_intersection3(...) равно 2140 миллисекунд  Время работы функции set\_intersection3(...) равно 2136 миллисекунд  Время работы функции set\_intersection3(...) равно 2143 миллисекунд  Математическое ожидание работы функции set\_intersection3(...) равно 2189  Дисперсия работы функции функции set\_intersection3(...) равно 1989  Доверительный интервал set\_intersection3(...) равно 2157 - 2220 |

**Подсчет вероятностных характеристик**

Для подсчета вероятностных характеристик в код был добавлен цикл на 10/50 запусков и получения времени исполнения, математического ожидания, дисперсии и доверительного интервала. Число 10 можно изменить до любого нужного значения.

Итого получаем:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Число потоков | Мат. Ожид | Дисперсия | Дов. инт. 0.95% |
| simple | 1  1 (50) | 3796  3700 | 4710  2588 | [3748 - 3843]  [3685 - 3714] |
| pthread | 1  1 (50)  2  2 (50)  4  4 (50)  8  8 (50) | 3771  3693  2027  1952  1646  1610  1667  1640 | 7956  2047  4709  384  908  451  663  505 | [3708 - 3833]  [3680 - 3705]  [1979 - 2074]  [1946 - 1957]  [1624 - 1667]  [1604 - 1615]  [1649 - 1684]  [1633 - 1646] |
| openmp | 1  1 (50)  2  2 (50)  4  4 (50)  8  8 (50) | 3737  3669  2189  2146  1744  1792  1659  1666 | 5096  3116  1989  202  3154  5833  749  5605 | [3686 - 3787]  [3653 - 3684]  [2157 - 2220]  [2142 - 2149]  [1703 - 1784]  [1770 - 1813]  [1639 - 1678]  [1644 - 1687] |

Из данной таблицы видно, что многопоточные приложения выигрывают по скорости выполнения у однопоточных. Библиотека pthread и openmp показывают приблизительно одинаковые результаты. Увеличение до 8 потоков ничего не дало, так как в системе 4 логических потока.

Кроме этого данные результаты зависят от загруженности системы, а это влияет на скорость выполнения программ.

**Вывод**

В ходе работы было создано 3 программы на языке С++ для решения задачи пересечения трех множеств. В ходе работы изучены и использованы библиотеки pthread и openmp. Реализованные решения на их основе позволяют легко менять количество исполняемых потоков. Эффективность библиотеки pthread эквивалента эффективности библиотеки openmp. Также библиотека OpenMP рассчитана на выполнение на одном компьютере в отличии от MPI.

Использование OpenMP оказалось очень удобным, так как логика программы не изменяется, просто добавляется пару строк в моем случае.