Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №8 по курсу «Объектно-ориентированное программирование»

Студент: Болотин А.Н.
Преподаватель: Поповкин А. В.
Группа: 08-207

Вариант: 19
Дата:
Оценка:
Подпись:

Москва, 2017

Лабораторная работа №8

1 Цель работы

• Знакомство с параллельным программированием в C++.

2 Задача

Используя структуры данных, разработанные для лабораторной работы №6 (контейнер 1-ого уровня и классы-фигуры) разработать алгоритм быстрой сортировки для класс-контейнера.

Необходимо разработать два вида алгоритма:
1. Обычный, без параллельных вызовов.
2. С использованием параллельных вызовов. В этом случае, каждый рекурсивный вызов сортировки должен создаваться в отдельном потоке.

Для создания потоков ипользовать механизмы:

- future
- packaged task/async

Для обеспечения потокобезопасности структур использовать механизмы:

- mutex
- lock quard

Программа должна позволять:

- Вводить произвольное количество фигур и добавлять их в контейнер.
- Распечатывать содержимое контейнера.
- Удалять фигуры из контейнера.
- Проводить сортировку контейнера.

Фигуры: Прямоугольник, ромб, трапеция
Контейнер 1: N-дерево **Контейнер 2:** Массив

3 Описание

Параллельное программирование – это техника программирования, которая использует преимущества многоядерных или многопроцесорных компьютеров и является подмножеством более широкого понятия многопоточности (multithreading).
Параллельное программирование может быть сложным, но его легче понять, если считать его не “трудным“ , а просто “немного иным“. Оно включает в себя все черты более традиционного, последовательного программирования, но в параллельном прогаммировании имеются три дополнительных, четко определенных этапа:

- Определение параллелизма: анализ задачи с целью выделить подзадачи, которые могут выполняться одновременно.
- Выявление параллелизма: изменение структуры задачи таким образом, чтобы можно было эффективно выполнять подзадачи. Для этого часто требуется найти зависимости между подзадачами и организовать исходный код так, чтобы ими можно было эффективно управлять.
- Выражение параллелизма: реализация параллельного алгоритма в исходном коде с помощью системы обозначений параллельного программирования.

4 Исходный код

```
template<class T>
size_t Tree<T>::partition(size_t arr[], size_t low, size_t high)
{
  int pivot = arr[high]; // pivot
  int i = (low - 1); // Index of smaller element

  for (int j = low; high != 0 && j <= high - 1; j++)
  {
    // If current element is smaller than or
    // equal to pivot
    if (arr[j] <= pivot)
    {
      i++; // increment index of smaller element
      //swap(&arr[i], &arr[j]);
      size_t tmp = arr[i];
      arr[i] = arr[j];
      arr[j] = tmp;
    }
  }
  //swap(&arr[i + 1], &arr[high]);
  size_t tmp = arr[i + 1];
```

```
  arr[i + 1] = arr[high];
  arr[high] = tmp;
  return (i + 1);
}

template<class T>
void Tree<T>::quickSort(size_t arr[], size_t low, size_t high)
{
  if (low < high)
  {

    size_t pi = partition(arr, low, high);
    if (pi > 0)
      quickSort(arr, low, pi - 1);
    quickSort(arr, pi + 1, high);
  }
}

size_t FutureQuickSort(size_t *arr, size_t left, size_t right)
{
  size_t i = left, j = right;

  size_t tmp;

  size_t pivot = arr[(left + right) / 2];

  while (i <= j) {
    while (arr[i] < pivot)
      i++;
    while (arr[j] > pivot)
      j--;
    if (i <= j) {
      tmp = arr[i];
      arr[i] = arr[j];
      arr[j] = tmp;
      i++;
      j--;
    }
  }

  if (i < right) {
    std::packaged_task<size_t(size_t *, size_t, size_t)> task(FutureQuickSort);
    auto result = task.get_future();

    std::thread task_td(std::move(task), std::ref(arr), i, right);
    task_td.join();
    result.get();
  }
  if (left < j) {
```

```
71      std::packaged_task<size_t(size_t *, size_t, size_t)> task(FutureQuickSort);
72      auto result = task.get_future();
73
74      std::thread task_td(std::move(task), std::ref(arr), left, j);
75      task_td.join();
76      result.get();
77    }
78    return 0;
79  }
```

5 Выводы

Благодаря этой лабораторной работе я получил опыт применения потоков в C++, что, несомненно, пригодится в дальнейшем. Распараллеливание вычислений - незаменимая вещь для уменьшения времени работы над каким-то алгоритмом, например сортировки работают в разы быстрее, чем в однопоточной реализации.