Отчёт по лабораторной работе № 2
Жуков Вадим, ИВТ-12М

Используемое железо: 2 физических ядра, 4 логических, 3.4ГГц

ОС: Windows 10

VS: Visual Studio 2017 v15.9.18

IPS: v2019

1. Разберите пример программы нахождения максимального элемента массива и его индекса task_for_lecture2.cpp. Запустите программу и убедитесь в корректности ее работы.

2. По аналогии с функцией *ReducerMaxTest(...)*, реализуйте функцию *ReducerMinTest(...)* для нахождения минимального элемента массива и его индекса. Вызовите функцию *ReducerMinTest(...)* до сортировки исходного массива *mass* и после сортировки. Убедитесь в правильности работы функции *ParallelSort(...)*: индекс минимального элемента после сортировки должен быть равен **0**, индекс максимального элемента (*mass_size* - **1**).

```
Task 2

Min = 3, index = 1184
Max = 24998, index = 4570
Min = 3, index = 0
Max = 24998, index = 9999
```

Немного изменил вывод и добавил функцию ReducerMinTest. Как видно в результатах, минимальный и максимальный элементы находятся в начале и конце соответственно, значит сортировка работает. В зависимости от задания я добавил соответствующую приписку каждой функции, чтобы можно было запустить всё в одном файле.

3. Добавьте в функцию ***ParallelSort(...)*** строки кода для измерения времени, необходимого для сортировки исходного массива. Увеличьте количество элементов ***mass_size*** исходного массива ***mass*** в **10**, **50**, **100** раз по сравнению с первоначальным. Выводите в консоль время, затраченное на сортировку массива, для каждого из значений ***mass_size***. ***Рекомендуется*** засекать время с помощью библиотеки ***chrono***.

```
Task 3

Array size: 100000
Before sorting
Min = 1, index = 39645
Max = 25000, index = 75652
After sorting
Min = 1, index = 0
Max = 25000, index = 99999
Sorting duration: 0.173102

Array size: 500000
Before sorting
Min = 1, index = 748
Max = 25000, index = 46954
After sorting
Min = 1, index = 0
Max = 25000, index = 499982
Sorting duration: 0.858746

Array size: 1000000
Before sorting
Min = 1, index = 4621
Max = 25000, index = 17929
After sorting
Min = 1, index = 0
Max = 25000, index = 999977
Sorting duration: 1.30883
```

Добавил замер времени, за которое производится сортировка, и измерил её для различных размерностей массива. Как видно по результатам, сортировка работает корректно: минимальный и максимальный элементы находятся на правильных местах и время сортировки увеличивается при увеличении размерности массива.

4. Реализуйте функцию ***CompareForAndCilk_For(size_t sz)***. Эта функция должна выводить на консоль время работы стандартного цикла ***for***, в котором заполняется случайными значениями ***std::vector*** (использовать функцию ***push_back(rand() % 20000 + 1)***), и время работы параллельного цикла ***cilk_for*** от ***Intel Cilk Plus***, в котором заполняется случайными значениями ***reducer вектор***.

Пример объявления reducer вектора: *cilk::reducer<cilk::op_vector<int>>red_vec;*

Пример его заполнения: *red_vec->push_back(rand() % 20000 + 1);*

Параметр функции ***sz*** - количество элементов в каждом из векторов.

Вызывайте функцию ***CompareForAndCilk_For()*** для входного параметра ***sz*** равного: **1000000**, **100000**, **10000**, **1000**, **500**, **100**, **50**, **10**. Проанализируйте результаты измерения времени, необходимого на заполнение ***std::vector***'а и ***reducer*** вектора.

```
Task 4

Array size: 1000000
Vector with for duration: 0.738069
Reducer with cilk_for duration: 0.323607

Array size: 100000
Vector with for duration: 0.0630626
Reducer with cilk_for duration: 0.035935

Array size: 10000
Vector with for duration: 0.00619478
Reducer with cilk_for duration: 0.00317799

Array size: 1000
Vector with for duration: 0.00175924
Reducer with cilk_for duration: 0.000923693

Array size: 500
Vector with for duration: 0.000300653
Reducer with cilk_for duration: 0.000416568

Array size: 100
Vector with for duration: 7.6371e-05
Reducer with cilk_for duration: 0.000190775

Array size: 50
Vector with for duration: 8.2106e-05
Reducer with cilk_for duration: 0.000219452

Array size: 10
Vector with for duration: 3.743e-05
Reducer with cilk_for duration: 5.8561e-05
```

На картинке представлено заполнение вектора с использованием стандартного for и заполнение reducer вектора с использованием cilk_for. Как видно, при маленьких размерностях массива (10, 50, 100, 500) стандартный for отрабатывает быстрее, при больших (1000 и больше) – cilk_for оказывается эффективнее.

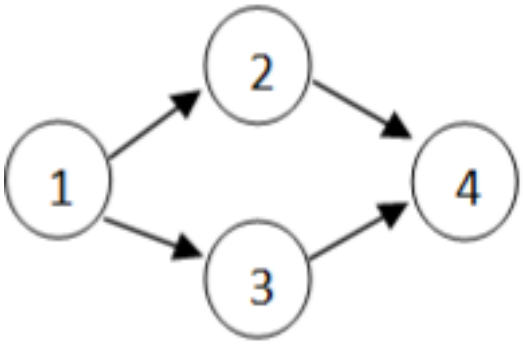
5. Ответьте на вопросы: почему при небольших значениях ***sz*** цикл ***cilk_for*** уступает циклу ***for*** в быстродействии? В каких случаях целесообразно использовать цикл ***cilk_for*** ? В чем принципиальное отличие параллелизации с использованием ***cilk_for*** от параллелизации с использованием ***cilk_spawn*** в паре с ***cilk_sync***?

cilk_for помимо вычисления также тратит время на создание потоков, распределение задач между ними и переключение контекста, которое сопоставимо с временем работы стандартного for при малых размерностях массива, поэтому получилось такое расхождение в 4 задании. Его использование будет эффективным при достаточно большом размере массива. Как показала практика, достаточно большая размерность для применения cilk_for – 1000 элементов и больше.

cilk_spawn: используется для указания, что данная функция может вызываться параллельно с вызывающей.

cilk_sync: используется для синхронизации результатов родительской функции и дочерней.

На рисунке функции 2 и 3 вызываются параллельно, функция 4 начинает выполнение только по окончании работы предыдущих двух.



cilk_for: используется для распараллеливания циклов с известным количеством повторений. В процессе компиляции тело цикла конвертируется в функцию, которая вызывается рекурсивно. Планировщик автоматически распределяет поддеревья рекурсии между обработчиками. На рисунке ниже показан принцип его работы.