Отчёт по лабораторной работе №2

Жуков Вадим, ИВТ-12М

Технические характеристики

OС: Windows 10

VS: Visual Studio 2017 v15.9.18

IPS: v2019

Железо: 2 физических ядра, 4 логических, 3.4ГГц

# Задание 1

Разберите пример программы нахождения максимального элемента массива и его индекса[**task\_for\_lecture2.cpp**](https://canvas.instructure.com/courses/1844563/files/87817220/download?wrap=1)**[Предварительный просмотр документа](https://canvas.instructure.com/courses/1844563/files/87817220/download?wrap=1).**Запустите программу и убедитесь в корректности ее работы.

Разобрал.

# Задание 2

По аналогии с функцией ***ReducerMaxTest(…)***, реализуйте ***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).**

Я немного изменил вывод и добавил функцию ReducerMinTest. Как видно, минимальный и максимальный элементы находятся в начале и конце соответственно, значит сортировка работает. Результат показан на рисунке 1.

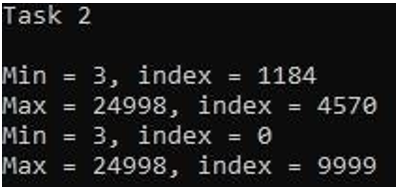


Рисунок 1. Задание 2, результат

# Задание 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**.

Результаты показаны на рисунке 2. Как видно, сортировка работает корректно:минимальный и максимальный элементы находятся в правильных местах и время сортировки увеличивается при увеличении размерности массива.

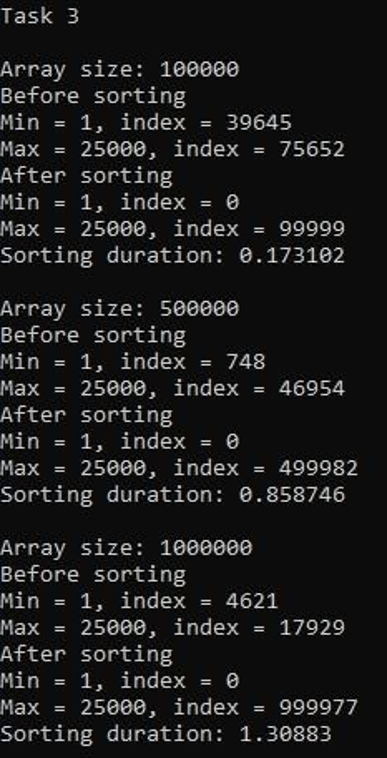


Рисунок 2. Задание 3, результат

# Задание 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**.

Результаты представлены на рисунке 3. Как видно, при маленьких размерностях массива (10, 50, 100, 500) стандартный for отрабатывает быстрее, при больших (1000 и больше) – cilk\_for оказывается эффективнее.

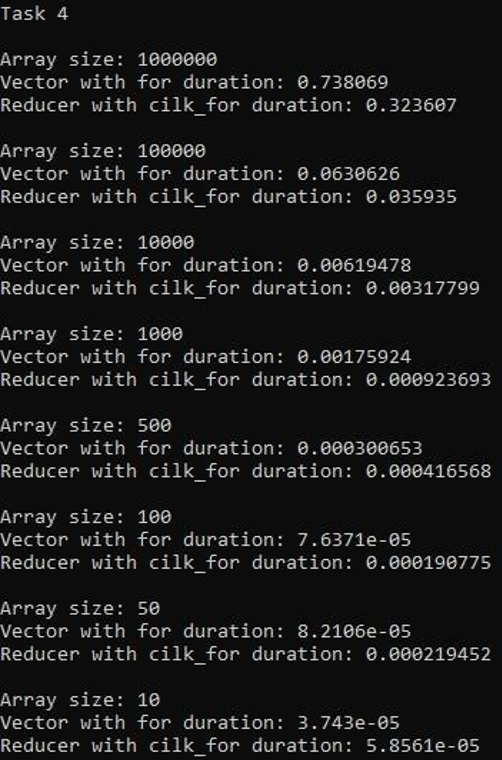


Рисунок 3. Задание 4, результат

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

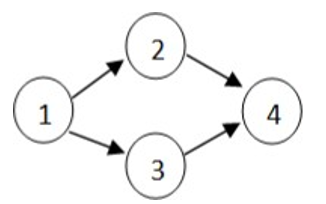


Рисунок 4. Задание 5, функции

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 5 показан принцип его работы.

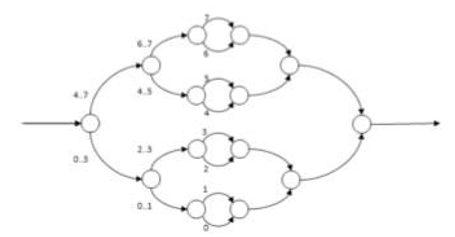


Рисунок 5. Задание 5, рекурсия