

Задание к занятию 2

Выполнил: Селиверстов Михаил

Группа: ИВТ-12М

Все необходимые программные коды к рисункам находятся на GitHub:

https://github.com/MichilMIET/exercise_2.git

```
--Before sort--
Maximal element = 24999 has index = 1098
Minimum element = 4 has index = 1530

--After sort--
Maximal element = 24999 has index = 9999
Minimum element = 4 has index = 0
```

Рисунок 1 - Поиск минимума и вывод его индекса 10000 размер массива (задание 2)

```
Mass size: 100000
Duration is: 0.118544 seconds
Mass size: 500000
Duration is: 0.696415 seconds
Mass size: 1000000
Duration is: 1.61532 seconds
```

Рисунок 2 - Подсчет времени работы функции ParallelSort(...) (задание 3)

```
Size mass: 1000000
Duration for normal FOR is: 0.841438 seconds
Duration for cilk_for FOR is: 0.331019 seconds

Size mass: 100000
Duration for normal FOR is: 0.0782325 seconds
Duration for cilk_for FOR is: 0.0333035 seconds

Size mass: 10000
Duration for normal FOR is: 0.00876309 seconds
Duration for cilk_for FOR is: 0.00907924 seconds

Size mass: 1000
Duration for normal FOR is: 0.000760301 seconds
Duration for cilk_for FOR is: 0.000708676 seconds

Size mass: 500
Duration for normal FOR is: 0.000418549 seconds
Duration for cilk_for FOR is: 0.000399777 seconds

Size mass: 100
Duration for normal FOR is: 0.000261967 seconds
Duration for cilk_for FOR is: 0.000198822 seconds

Size mass: 50
Duration for normal FOR is: 6.9545e-05 seconds
Duration for cilk_for FOR is: 0.000207782 seconds

Size mass: 10
Duration for normal FOR is: 2.9013e-05 seconds
Duration for cilk_for FOR is: 0.000349858 seconds
```

Рисунок 3 - Сравнение скорости работы for и cilk_for при различных размерах вектора (задание 4)

Ответы на вопросы 5 задания

1. Почему при небольших значениях `sz` цикл `cilk_for` уступает циклу `for` в быстродействии?

Поскольку в цикле `cilk_for` рекурсивная функция разделяет задачу вычислений на два блока условно говоря, (более подробно во втором вопросе отражено и на рисунке 4) и соответственно при малых значениях `sz` времени на разделение в `cilk_for` тратиться куда больше, чем при работе обычного цикла `for`.

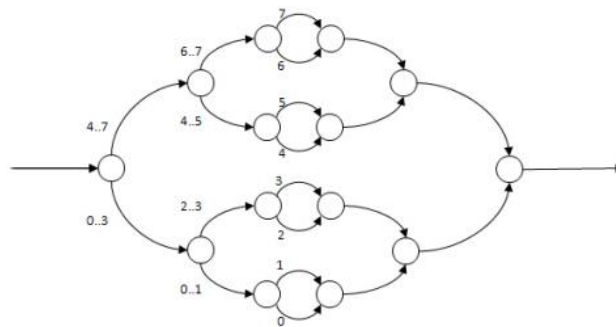


Рисунок 4 - Диаграмма создания нитей при распараллеливании цикла с известным числом итераций с помощью конструкции `cilk_for`

2. В каких случаях целесообразно использовать цикл `cilk_for`?

Цикл `cilk_for` работает по следующему принципу: компилятор преобразует тело цикла в функцию, которая вызывается рекурсивно, на каждом уровне рекурсии половина оставшейся работы выполняется потомком, а вторая половина – продолжением, что отражено на рисунке 4. Соответственно можно применять данный цикл если у нас итерации не зависимы по данным, так же у этого цикла есть ряд ограничений:

- Распараллеливаются только циклы без цикловых зависимостей (итерации могут выполняться независимо).
- Недопустимы переходы в тело цикла и из него (операторы `return`, `break`, `goto` с переходом из тела цикла или в тело цикла).
- В цикле должна быть только одна переменная цикла.
- Переменная цикла не должна модифицироваться в цикле.
- Границы изменения параметра и шаг не должны меняться в теле цикла.
- Цикл не должен быть бесконечным.

3. В чем принципиальное отличие параллелизации с использованием `cilk_for` от параллелизации с использованием `cilk_spawn` в паре с `cilk_sync`?

Используя конструкцию `cilk_spawn`, можно “распараллелить” вычисления, которые требуют зависимости по данным, а именно можно “ждать” результатов промежуточных вычислений, используя конструкцию `cilk_sync`, соответственно в этом плане и отличие от цикла `cilk_for` который этого не позволяет.