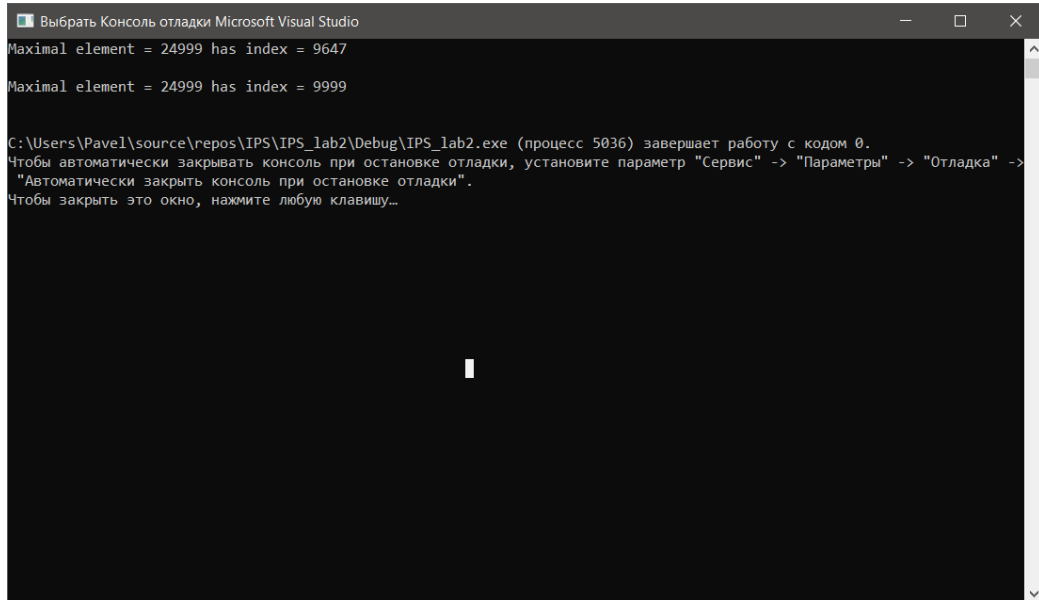


**Программное обеспечение, использованное при выполнении:** Visual Studio 2017, IPS 2019, Windows 10 – 64x

**Процессор** – четырехъядерный Intel Core i5 8250U с частотой 1.6 ГГц

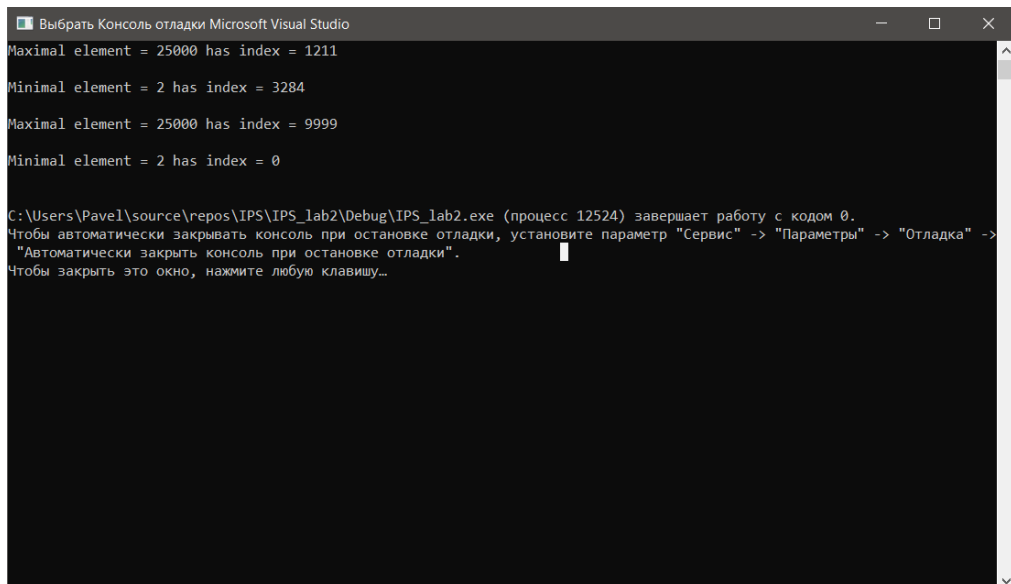
1. Разберите пример программы нахождения максимального элемента массива и его индекса task\_for\_lecture2.cpp. Предварительный просмотр документа. Запустите программу и убедитесь в корректности ее работы.



```
Выбрать Консоль отладки Microsoft Visual Studio
Maximal element = 24999 has index = 9647
Maximal element = 24999 has index = 9999
C:\Users\Pavel\source\repos\IPS\IPS_lab2\Debug\IPS_lab2.exe (процесс 5036) завершает работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, установите параметр "Сервис" -> "Параметры" -> "Отладка" ->
"Автоматически закрывать консоль при остановке отладки".
Чтобы закрыть это окно, нажмите любую клавишу...
```

Рисунок 1 – пример работы программы

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).



```
Выбрать Консоль отладки Microsoft Visual Studio
Maximal element = 25000 has index = 1211
Minimal element = 2 has index = 3284
Maximal element = 25000 has index = 9999
Minimal element = 2 has index = 0
C:\Users\Pavel\source\repos\IPS\IPS_lab2\Debug\IPS_lab2.exe (процесс 12524) завершает работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, установите параметр "Сервис" -> "Параметры" -> "Отладка" ->
"Автоматически закрывать консоль при остановке отладки".
Чтобы закрыть это окно, нажмите любую клавишу...
```

Рисунок 2 – пример работы программы

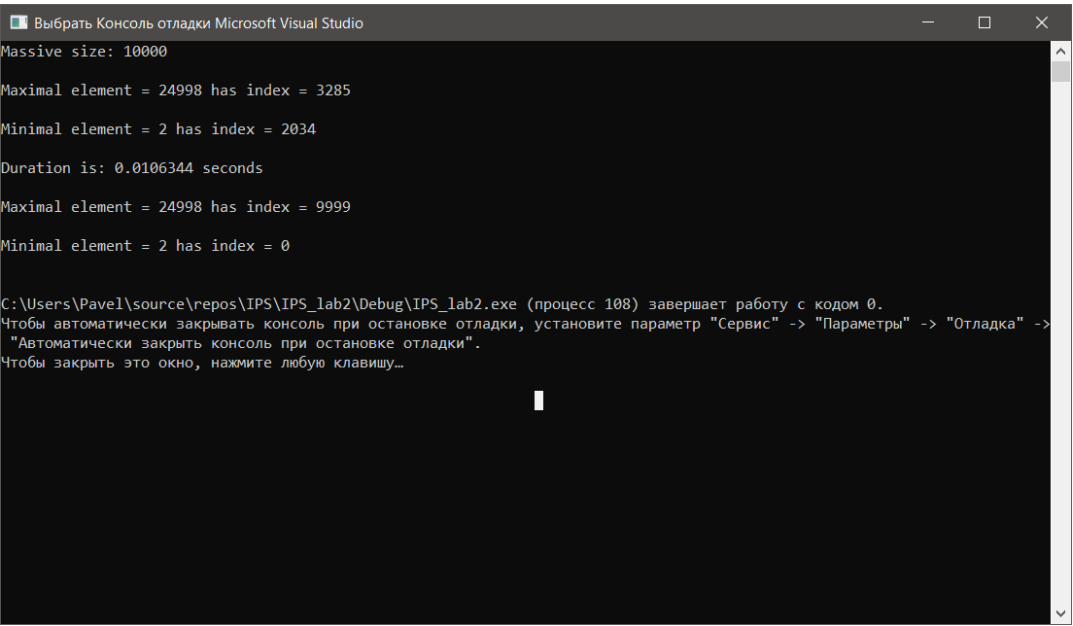
3. Добавьте в функцию ParallelSort(...) строки кода для измерения времени, необходимого для сортировки исходного массива. Увеличьте количество элементов mass\_size исходного массива

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.

В таблице, представлена зависимость времени, затраченного на выполнение программы, в зависимости от размера программы

Таблица 1 – зависимость времени выполнения от размера массива

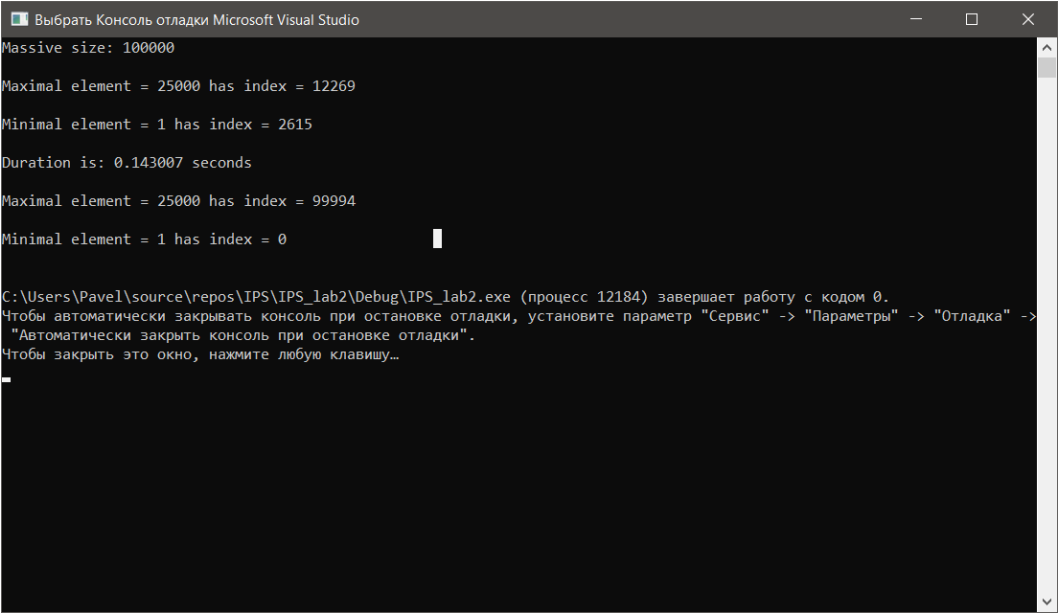
Размер массива	Время, с
10000	0.0106344
100000	0.143007
500000	0.710581
1000000	1.54245



```
Выбрать Консоль отладки Microsoft Visual Studio
Massive size: 10000
Maximal element = 24998 has index = 3285
Minimal element = 2 has index = 2034
Duration is: 0.0106344 seconds
Maximal element = 24998 has index = 9999
Minimal element = 2 has index = 0

C:\Users\Pavel\source\repos\IPS\IPS_lab2\Debug\IPS_lab2.exe (процесс 108) завершает работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, установите параметр "Сервис" -> "Параметры" -> "Отладка" ->
"Автоматически закрыть консоль при остановке отладки".
Чтобы закрыть это окно, нажмите любую клавишу...
```

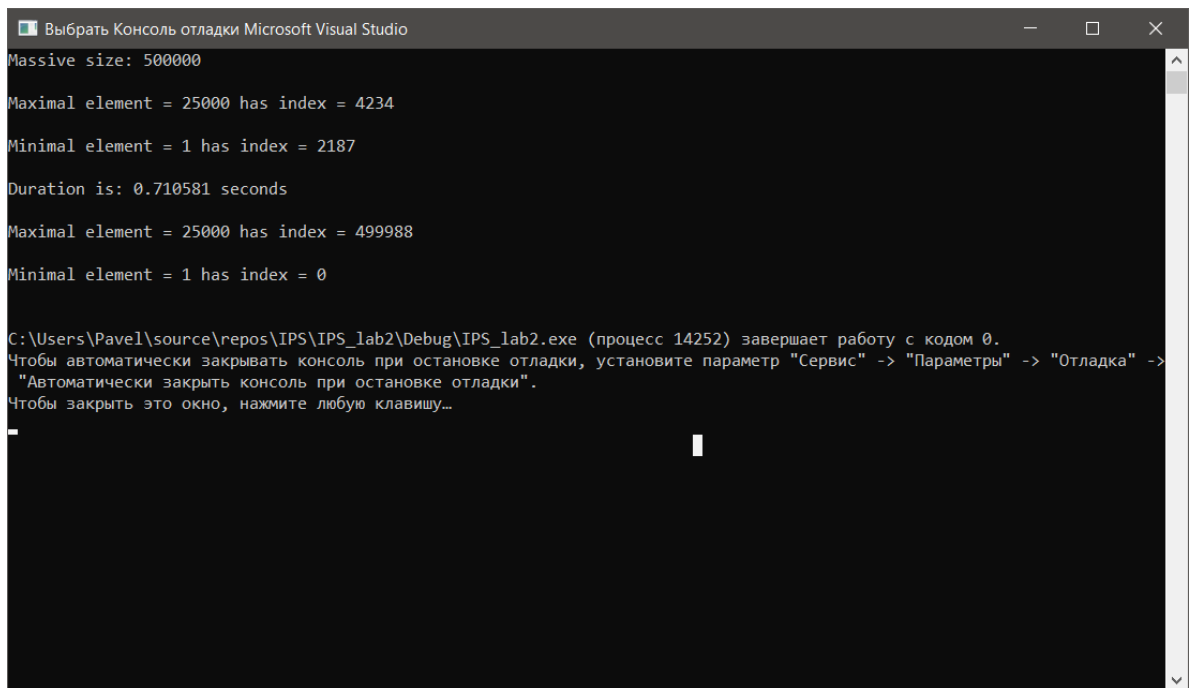
Рисунок 3 – пример работы программы при массиве из 10000 элементов



```
Выбрать Консоль отладки Microsoft Visual Studio
Massive size: 100000
Maximal element = 25000 has index = 12269
Minimal element = 1 has index = 2615
Duration is: 0.143007 seconds
Maximal element = 25000 has index = 99994
Minimal element = 1 has index = 0

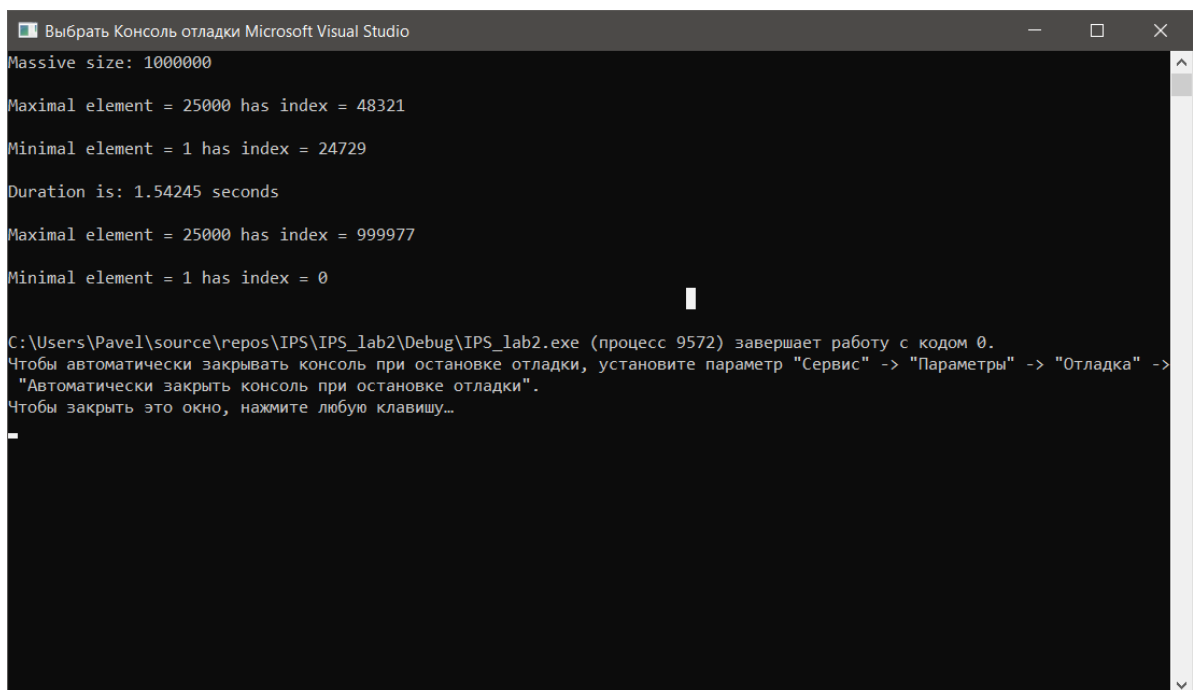
C:\Users\Pavel\source\repos\IPS\IPS_lab2\Debug\IPS_lab2.exe (процесс 12184) завершает работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, установите параметр "Сервис" -> "Параметры" -> "Отладка" ->
"Автоматически закрыть консоль при остановке отладки".
Чтобы закрыть это окно, нажмите любую клавишу...
```

Рисунок 4 – пример работы программы при массиве из 100000 элементов



```
Выбрать Консоль отладки Microsoft Visual Studio
Massive size: 500000
Maximal element = 25000 has index = 4234
Minimal element = 1 has index = 2187
Duration is: 0.710581 seconds
Maximal element = 25000 has index = 499988
Minimal element = 1 has index = 0
C:\Users\Pavel\source\repos\IPS\IPS_lab2\Debug\IPS_lab2.exe (процесс 14252) завершает работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, установите параметр "Сервис" -> "Параметры" -> "Отладка" ->
"Автоматически закрыть консоль при остановке отладки".
Чтобы закрыть это окно, нажмите любую клавишу...
```

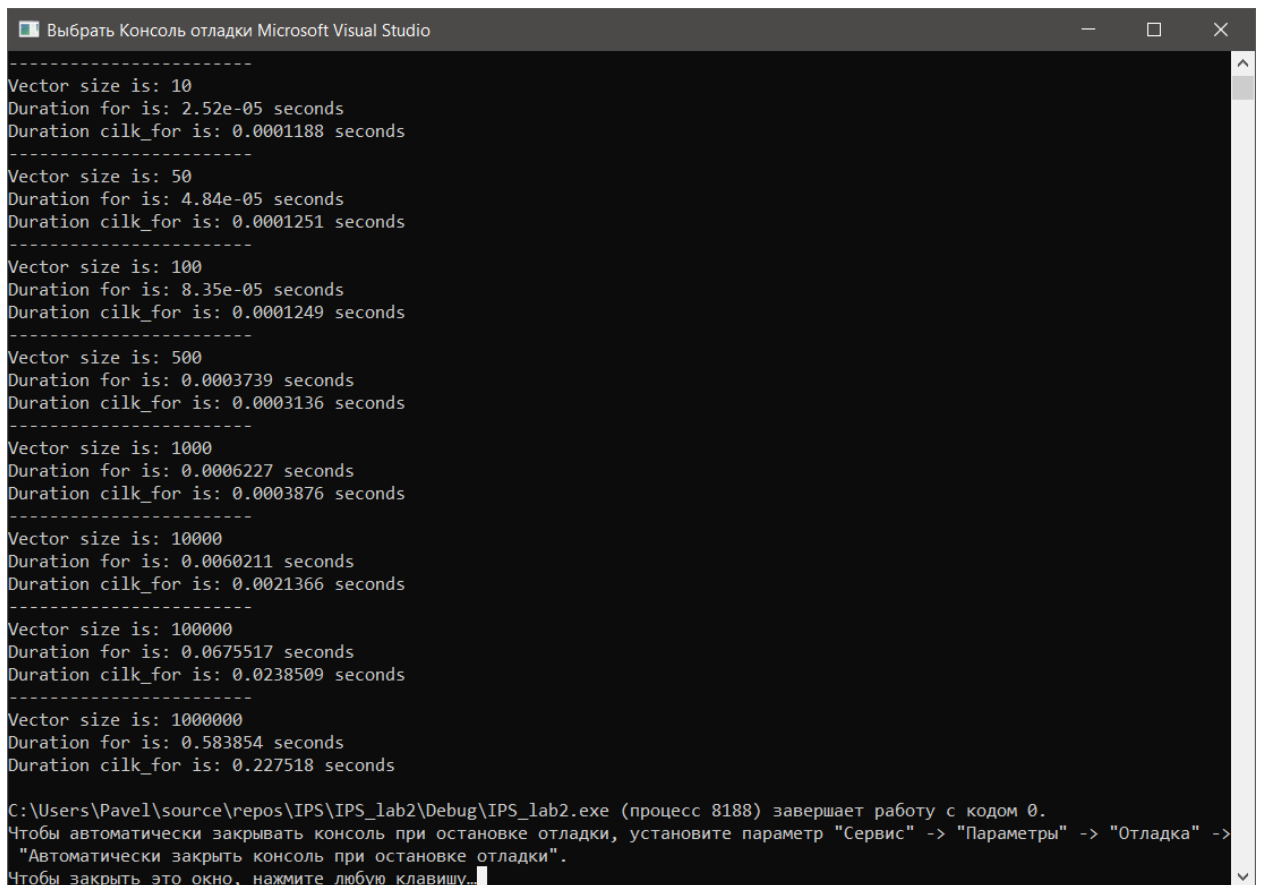
Рисунок 5 – пример работы программы при массиве из 500000 элементов



```
Выбрать Консоль отладки Microsoft Visual Studio
Massive size: 1000000
Maximal element = 25000 has index = 48321
Minimal element = 1 has index = 24729
Duration is: 1.54245 seconds
Maximal element = 25000 has index = 999977
Minimal element = 1 has index = 0
C:\Users\Pavel\source\repos\IPS\IPS_lab2\Debug\IPS_lab2.exe (процесс 9572) завершает работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, установите параметр "Сервис" -> "Параметры" -> "Отладка" ->
"Автоматически закрыть консоль при остановке отладки".
Чтобы закрыть это окно, нажмите любую клавишу...
```

Рисунок 6 – пример работы программы при массиве из 1000000 элементов

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.



```
-----
Vector size is: 10
Duration for is: 2.52e-05 seconds
Duration cilk_for is: 0.0001188 seconds
-----
Vector size is: 50
Duration for is: 4.84e-05 seconds
Duration cilk_for is: 0.0001251 seconds
-----
Vector size is: 100
Duration for is: 8.35e-05 seconds
Duration cilk_for is: 0.0001249 seconds
-----
Vector size is: 500
Duration for is: 0.0003739 seconds
Duration cilk_for is: 0.0003136 seconds
-----
Vector size is: 1000
Duration for is: 0.0006227 seconds
Duration cilk_for is: 0.0003876 seconds
-----
Vector size is: 10000
Duration for is: 0.0060211 seconds
Duration cilk_for is: 0.0021366 seconds
-----
Vector size is: 100000
Duration for is: 0.0675517 seconds
Duration cilk_for is: 0.0238509 seconds
-----
Vector size is: 1000000
Duration for is: 0.583854 seconds
Duration cilk_for is: 0.227518 seconds
-----
C:\Users\Pavel\source\repos\IPS\IPS_lab2\Debug\IPS_lab2.exe (процесс 8188) завершает работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, установите параметр "Сервис" -> "Параметры" -> "Отладка" ->
"Автоматически закрыть консоль при остановке отладки".
Чтобы закрыть это окно, нажмите любую клавишу.
```

Рисунок 7 – сравнение `cilk_for` и `for` в зависимости от размера массива

На рисунке видно, что при размерности вектора меньше 500, желательно использовать цикл `for`, при размерности массива данных больше 500, выгоднее использовать цикл `cilk_for`.

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***?

1) Почему при небольших значениях *sz* цикл ***cilk\_for*** уступает циклу ***for*** в быстродействии?

Потому что, операция создания потоков и распределения вычислительных задач между ними при использовании ***cilk\_for***, занимает некоторое время и в случае с небольшим размером массива данных, не имеет особого выигрыша по времени, а иногда даже на порядок выше.

2) В каких случаях целесообразно использовать цикл ***cilk\_for***?

`Cilk_for` гораздо выгоднее использовать с массивами данных большой размерности, на моем железе выигрыш в скорости идет уже при размерности больше 500. Соответственно, чем больше размерность, тем больше выигрыш в скорости выполнения. `Cilk_for` выгодно будет использовать в алгоритмах брутфорса, где нужно просто перебирать много данных.

3) В чем принципиальное отличие параллелизации с использованием ***cilk\_for*** от параллелизации с использованием ***cilk\_spawn*** в паре с ***cilk\_sync***?

Ключевое слово `cilk_spawn` сообщает компилятору, что функция, которой предшествует `cilk_spawn`, может работать параллельно с вызывающей функцией.

cilk\_sync – это точка синхронизации, которая заставляет, созданные ранее задачи, ждать завершения друг друга.

Пример использования, данных конструкций, приведен в таблице 2. В примере слово Done гарантировано выведется после того, как будут выполнены функции hello() и world()

Таблица 2 – пример использования cilk\_spawn и cilk\_sync

```
#include <iostream>
#include <cilk/cilk.h>
using namespace std;

int main(){
    cilk_spawn hello();
    cilk_spawn world();
    cilk_sync;
    cout << "Done! ";
}
```

Конструкция cilk\_for предназначена, для введения параллелизма в циклах for.

Четкая разница между cilk\_for и cilk\_spawn вместе с cilk\_sync видна на примере ниже:

```
for (int x = 0; x < n; ++x) { cilk_spawn f(x); }
cilk_for (int x = 0; x < n; ++x) { f(x); }
```

В первом случае, мы на каждой итерации будем создавать по задаче, а операция захвата чужой задачи весьма затратная с точки зрения производительности. Если в каждой итерации «мало» работы, то мы больше потеряем, чем получим с помощью такой «параллельной» программы.

**Вывод:** при выполнении задания, я познакомился с функцией **ReducerMinTest(...)**. Сравнил, затраченное на сортировку массива при разной размерности массива данных. Реализовал функцию **CompareForAndCilk\_For(size\_t sz)**, которая замеряет время, затраченное на выполнение циклов for и cilk\_for при разных размерностях массива. Также стало очевидным, что cilk\_for гораздо выгоднее использовать при больших размерах массива.