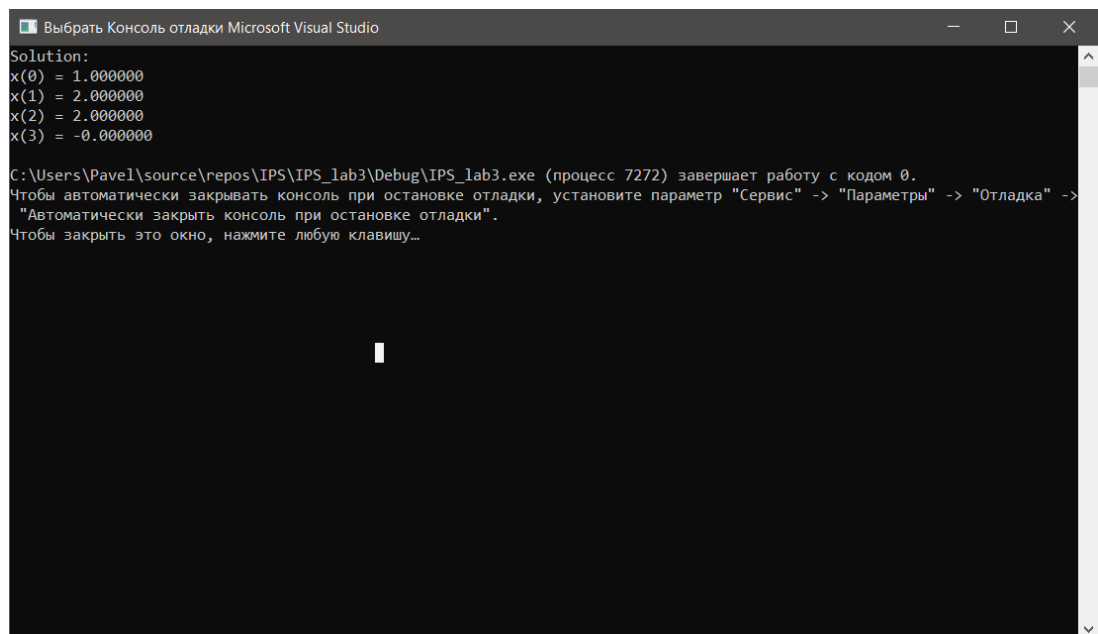


Программное обеспечение, использованное при выполнении: Visual Studio 2017, IPS 2019, Windows 10 – 64x

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

1. Запустите первоначальную версию программы и получите решение для тестовой матрицы **test_matrix**, убедитесь в правильности приведенного алгоритма. Добавьте строки кода для измерения времени (см. задание к занятию 2) выполнения прямого хода метода Гаусса в функцию **SerialGaussMethod()**. Заполните матрицу с количеством строк **MATRIX_SIZE** случайными значениями, используя функцию **InitMatrix()**. Найдите решение СЛАУ для этой матрицы (закомментируйте строки кода, где используется тестовая матрица **test_matrix**).

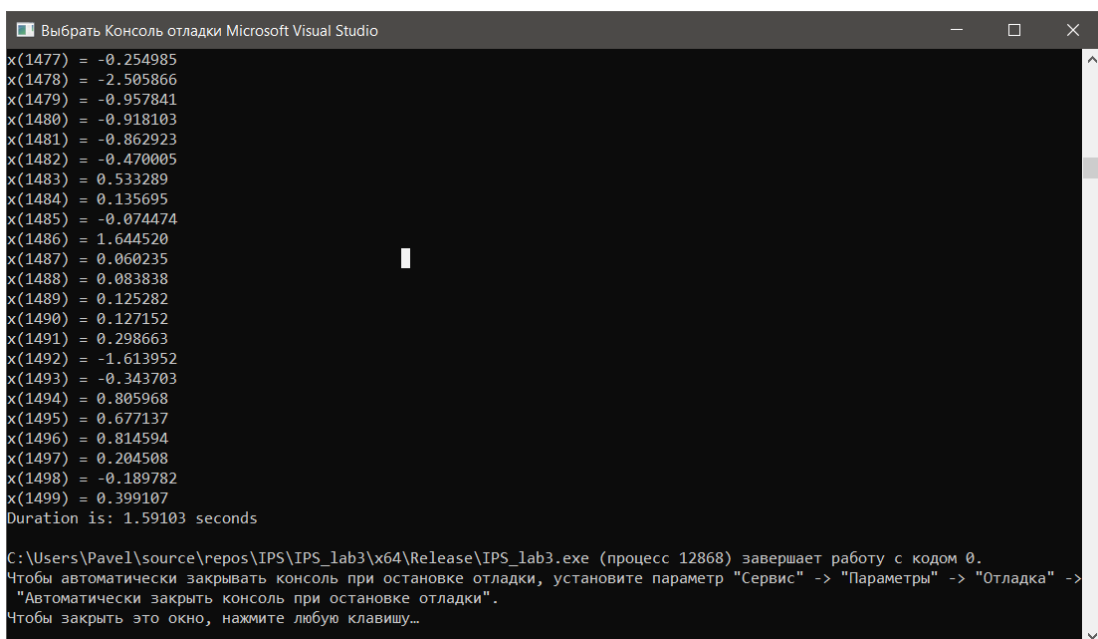


```
Выбрать Консоль отладки Microsoft Visual Studio

Solution:
x(0) = 1.000000
x(1) = 2.000000
x(2) = 2.000000
x(3) = -0.000000

C:\Users\Pavel\source\repos\IPS\IPS_lab3\Debug\IPS_lab3.exe (процесс 7272) 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 1 – Пример запуска программы для тестовой матрицы



```
Выбрать Консоль отладки Microsoft Visual Studio

x(1477) = -0.254985
x(1478) = -2.505866
x(1479) = -0.957841
x(1480) = -0.918103
x(1481) = -0.862923
x(1482) = -0.470005
x(1483) = 0.533289
x(1484) = 0.135695
x(1485) = -0.074474
x(1486) = 1.644520
x(1487) = 0.060235
x(1488) = 0.083838
x(1489) = 0.125282
x(1490) = 0.127152
x(1491) = 0.298663
x(1492) = -1.613952
x(1493) = -0.343703
x(1494) = 0.805968
x(1495) = 0.677137
x(1496) = 0.814594
x(1497) = 0.204508
x(1498) = -0.189782
x(1499) = 0.399107
Duration is: 1.59103 seconds

C:\Users\Pavel\source\repos\IPS\IPS_lab3\x64\Release\IPS_lab3.exe (процесс 12868) 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 2 – пример работы программы для матрицы размера 1500

2. С помощью инструмента **Amplifier XE** определите наиболее часто используемые участки кода новой версии программы. Сохраните скриншот результатов анализа **Amplifier XE**. Создайте, на основе последовательной функции **SerialGaussMethod()**, новую функцию, реализующую параллельный метод Гаусса. Введите параллелизм в новую функцию, используя **cilk_for**. **Примечание:** произвести параллелизацию одного внутреннего цикла прямого хода метода Гаусса (определить какого именно), и внутреннего цикла обратного хода. Время выполнения по-прежнему измерять только для прямого хода.

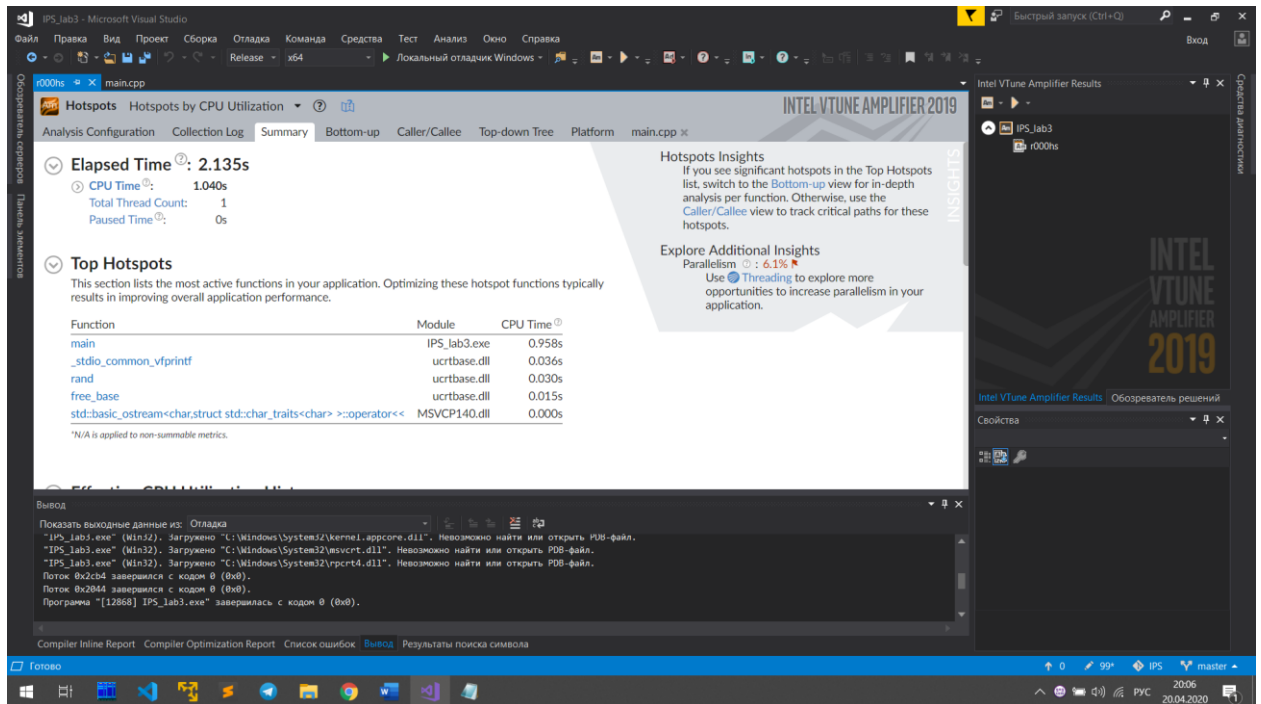


Рисунок 3 – Результаты работы Amplifier

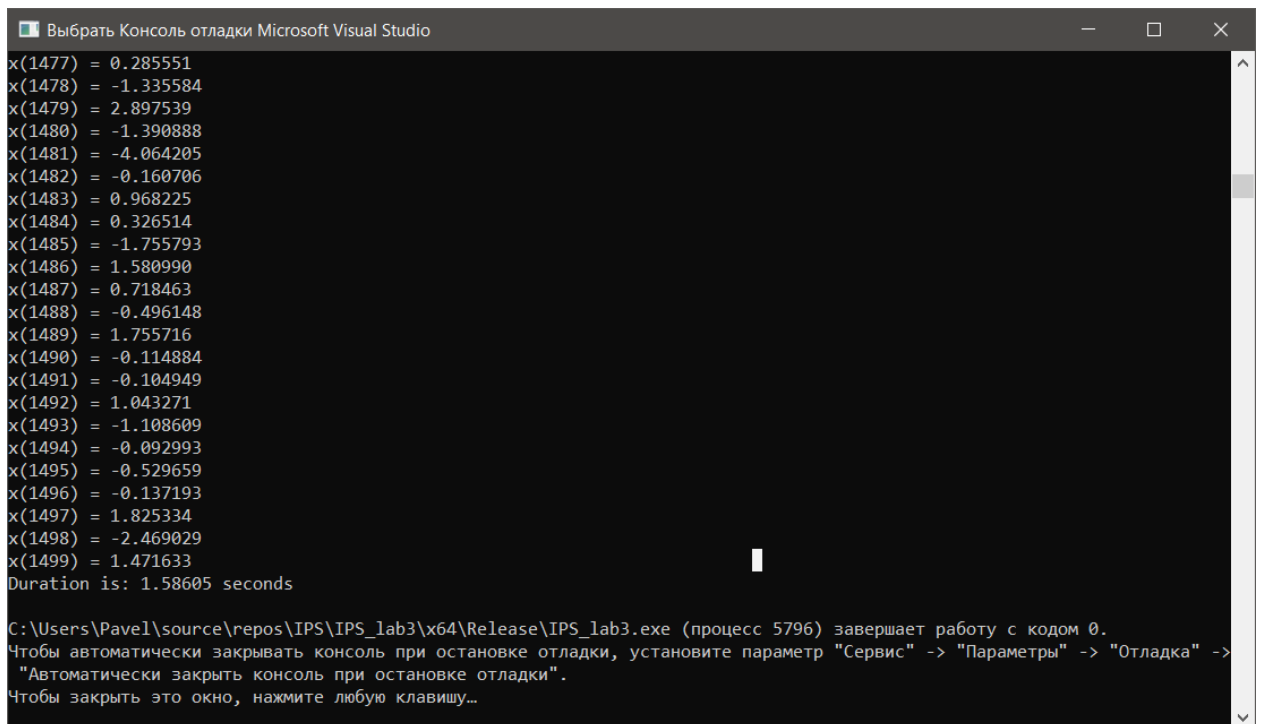


Рисунок 4 – Пример работы параллельного метода

3. Далее, используя **Inspector XE**, определите те данные (если таковые имеются), которые принимают участие в гонке данных или в других основных ошибках, возникающих при разработке параллельных программ, и устраните эти ошибки. Сохраните скриншоты анализов, проведенных инструментом **Inspector XE**: в случае обнаружения ошибок и после их устранения.

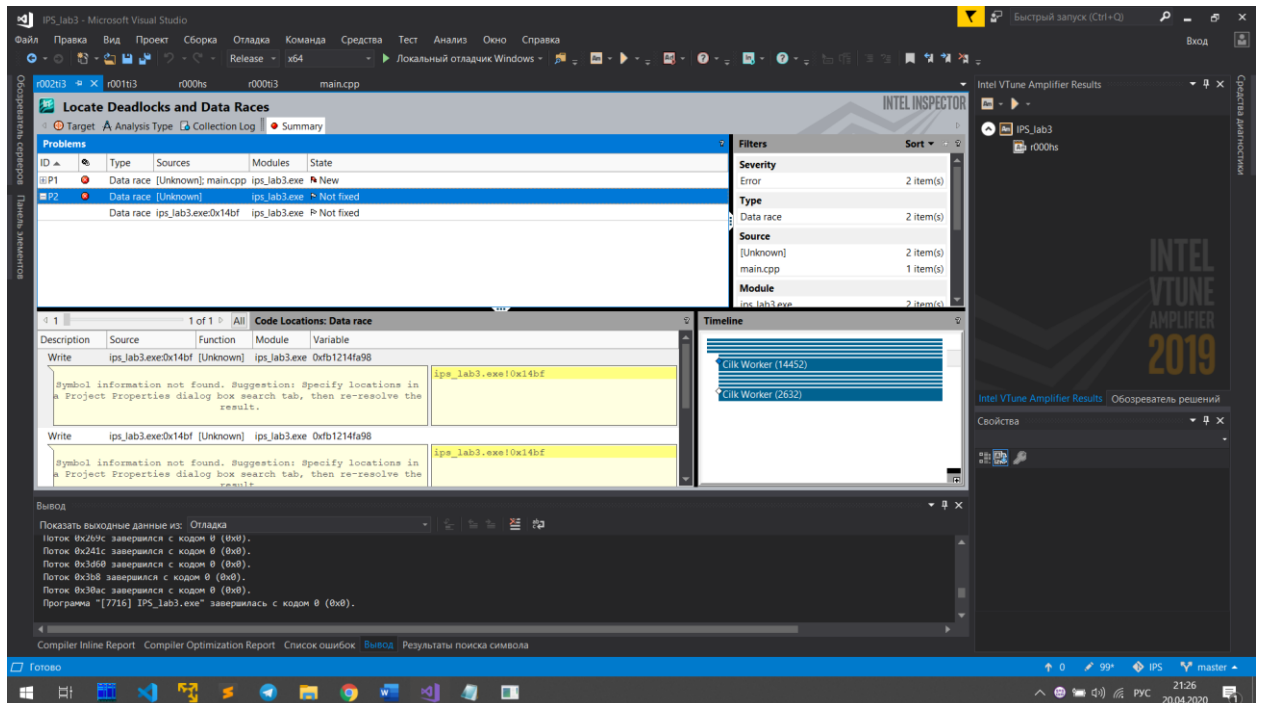


Рисунок 5 – Пример работы Inspector XE

Были найдены гонки данных, исправим их с помощью reducer.

Таблица 1 – Пример использования reducer_opadd

```
for(k = rows - 2; k >= 0; --k)
{
    cilk::reducer_opadd<double> result_k(matrix[k][rows]);

    //
    cilk_for (int j = k + 1; j < rows; ++j)
    {
        result[k] -= matrix[k][j] * result[j];
    }

    result[k] = result_k->get_value() / matrix[k][k];
}
```

Анализ после устранения ошибок

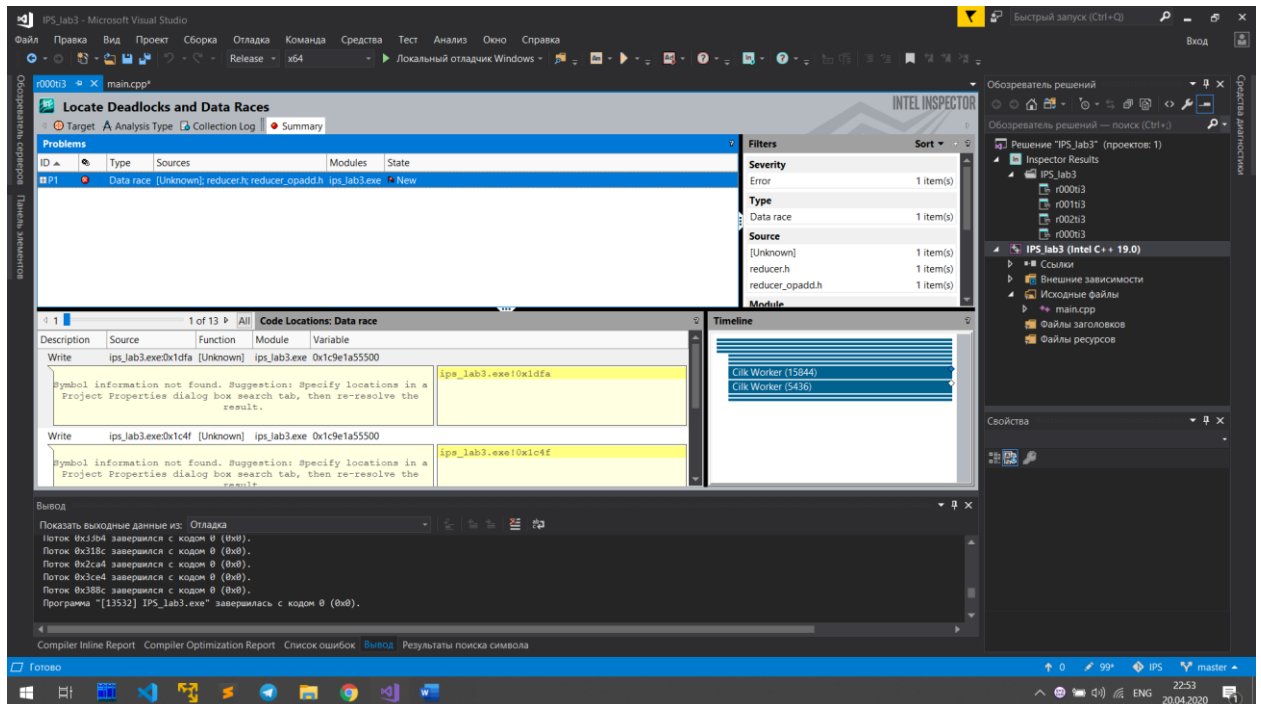


Рисунок 5 – Пример работы Inspector XE после добавления reducer

4. Убедитесь на примере тестовой матрицы **test_matrix** в том, что функция, реализующая параллельный метод Гаусса, работает правильно. Сравните время выполнения прямого хода метода Гаусса для последовательной и параллельной реализации при решении матрицы, имеющей количество строк **MATRIX_SIZE**, заполняющейся случайными числами. Запускайте проект в режиме **Release**, предварительно убедившись, что включена оптимизация (**Optimization**->**Optimization=O2**). Подсчитайте ускорение параллельной версии в сравнении с последовательной. Выводите значения ускорения на консоль.

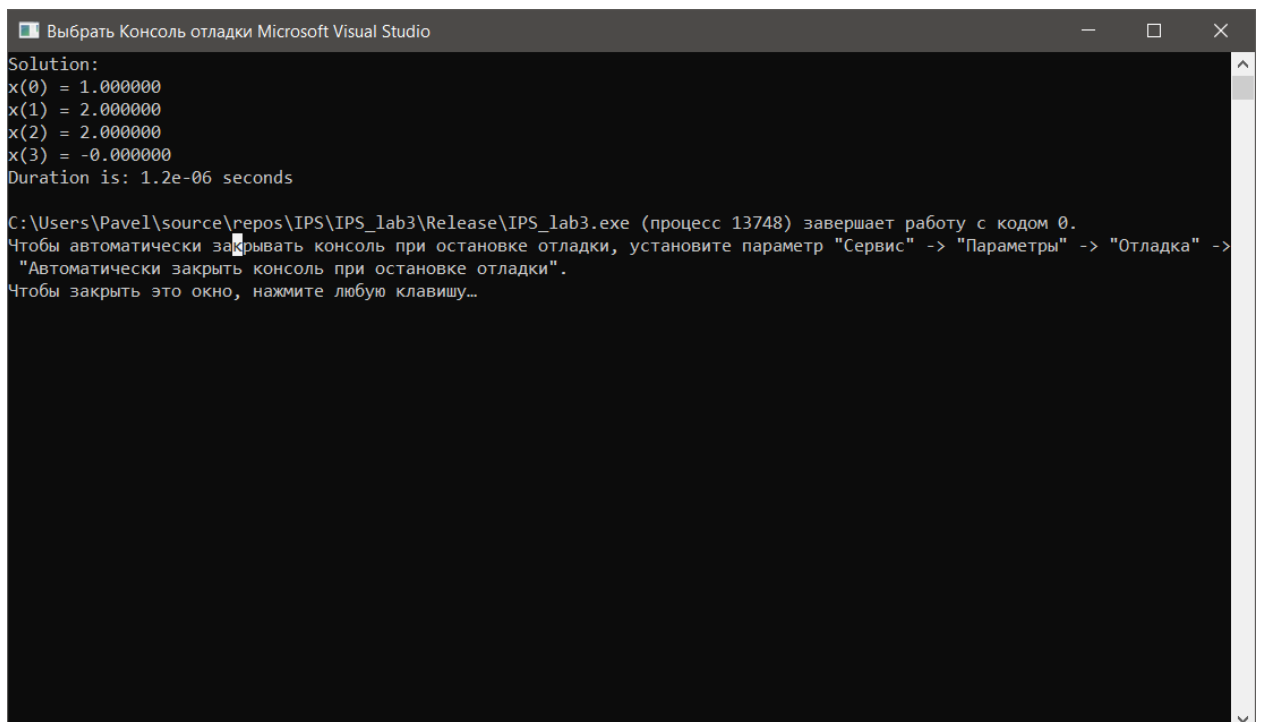
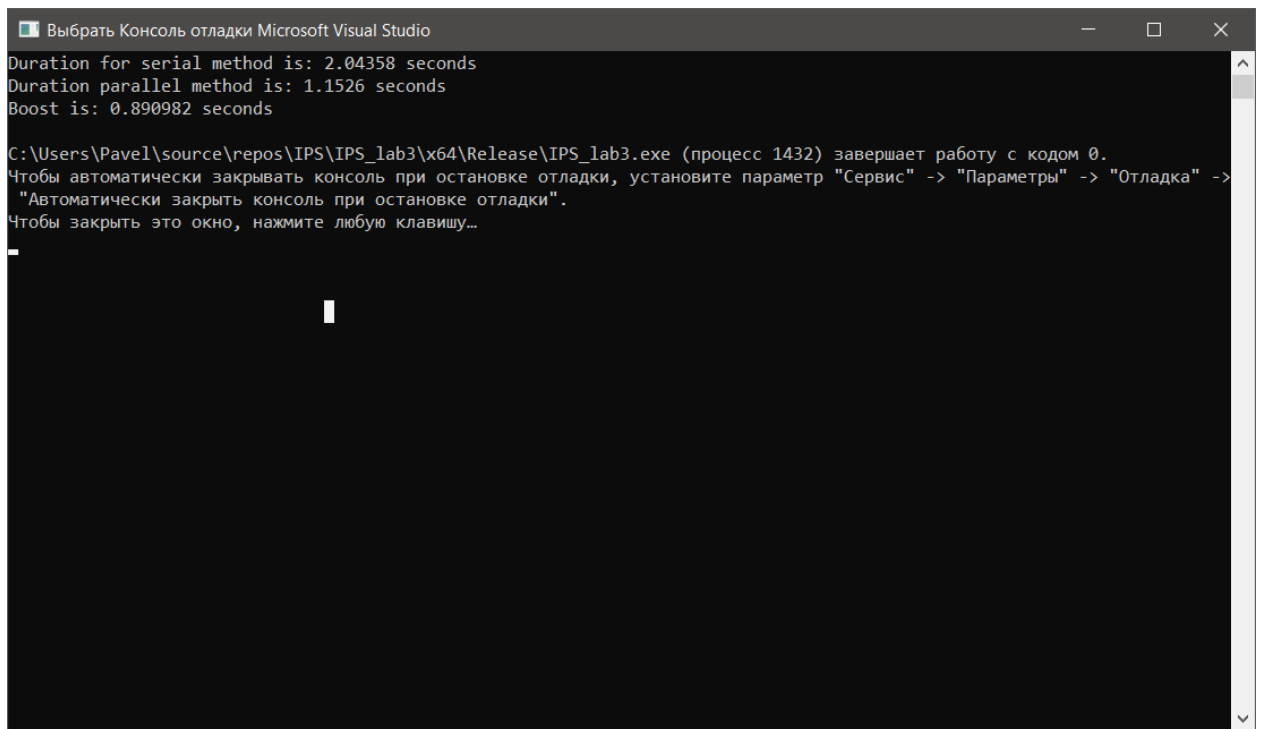


Рисунок 6 – пример работы параллельного метода для тестовой матрицы



```
Выбрать Консоль отладки Microsoft Visual Studio
Duration for serial method is: 2.04358 seconds
Duration parallel method is: 1.1526 seconds
Boost is: 0.890982 seconds

C:\Users\Pavel\source\repos\IPS\IPS_lab3\x64\Release\IPS_lab3.exe (процесс 1432) 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 последовательного и параллельного метода

Вывод: таким образом, при использовании `cilk_for` для достаточно большой размерности (1500) матрицы при решении СЛАУ методом Гаусса мы получили выигрыш примерно в 1 секунду. Также для того, чтобы получить корректные ответы и избежать гонок данных был использован `reducer_opadd`.