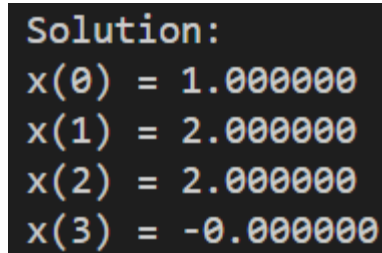


Отчёт по лабораторной работе №3

Петрова О.А. группы ИВТ-12М

1. В файле *task_for_lecture3.cpp* приведен код, реализующий последовательную версию метода Гаусса для решения СЛАУ. Проанализируйте представленную программу.



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

Рисунок 1. Результат выполнения программы из файла *task_for_lecture3.cpp*

2. 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*).

Примечание: в данной задаче (и последующих) обычные си-шные массивы с выделением и освобождением памяти заменены на вектора, т.к. у векторов не требуется задумываться о выделенной памяти.

```
x(1473) = 5.055432
x(1474) = 6.760372
x(1475) = 3.575086
x(1476) = 4.248012
x(1477) = 3.895500
x(1478) = 11.472871
x(1479) = 9.554044
x(1480) = -2.525995
x(1481) = -0.711334
x(1482) = 7.741069
x(1483) = -3.286473
x(1484) = -0.774482
x(1485) = 2.715142
x(1486) = -4.006153
x(1487) = 8.572835
x(1488) = 1.212041
x(1489) = -0.735586
x(1490) = 7.214756
x(1491) = 3.762978
x(1492) = 5.124497
x(1493) = 12.984607
x(1494) = 3.255437
x(1495) = -1.203453
x(1496) = 4.252704
x(1497) = 3.476689
x(1498) = 13.943574
x(1499) = -10.088711
Time: 1.44497
```

Рисунок 2. Результат вычисления СЛАУ для новой матрицы и время вычисления прямого хода метода Гаусса

3. 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.

Elapsed Time [Ⓜ]: 3.510s

CPU Time [Ⓜ]: 1.988s

Total Thread Count: 1

Paused Time [Ⓜ]: 0s

Top Hotspots [Ⓜ]

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time [Ⓜ]
SerialGaussMethod	Oleg.exe	1.732s
std::operator<<	Oleg.exe	0.110s
std::basic_ostream<char,struct std::char_traits<char> >::operator<<	MSVCP140.dll	0.044s
std::basic_ostream<char,struct std::char_traits<char> >::operator<<	MSVCP140.dll	0.039s
std::vector<std::vector<double, std::allocator<double> >, std::allocator<std::vector<double, std::allocator<double> > >::vector	Oleg.exe	0.020s
[Others]		0.042s

*N/A is applied to non-summable metrics.

Effective CPU Utilization Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the

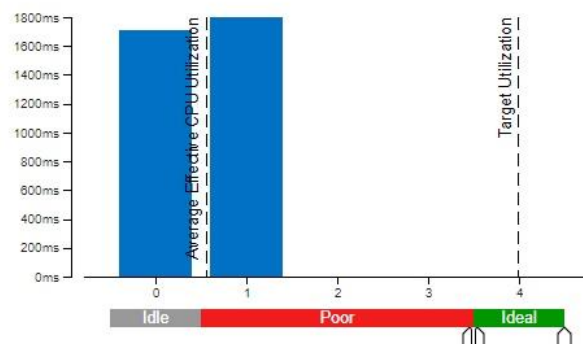


Рисунок 3. Результат поиска часто используемых участков кода с помощью инструмента Amplifier XE

В результате к поиску видно, что большую часть времени исполнения программа проводит в функции SerialGaussMethod.

Затем требовалось написать аналогичную функции SerialGaussMethod функцию ParallelGaussMethod, только уже с вложенными циклами cilk_for.

```
x(1499) = -0.0521182
Time: 2.6011737
```

Рисунок 4. Результат вычисления СЛАУ с помощью функции ParallelGaussMethod и время вычисления прямого хода метода Гаусса

4. 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.

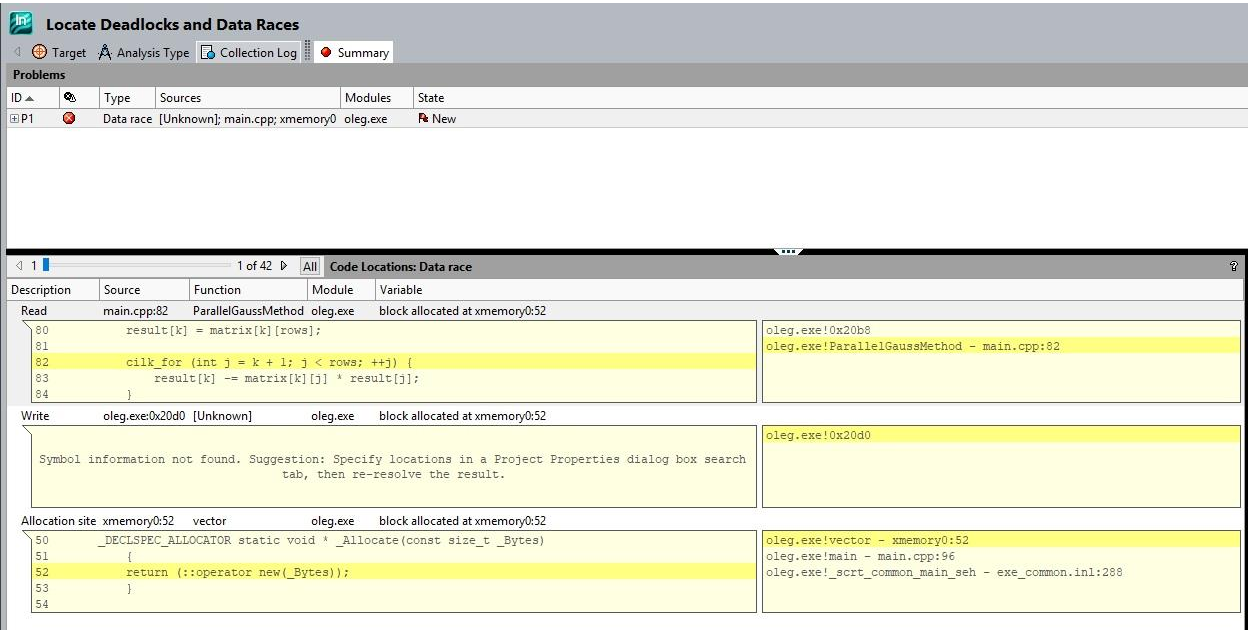


Рисунок 5. Результат Inspector XE с данными, которые принимают участие в гонке данных

Чтобы исправить обнаруженную гонку данных используем редьюсер реализованный в библиотеке *cilk/reducer_opadd.h*. Затем прогоним ещё раз программу через Instpector XE чтобы убедиться, что гонки данных больше нет.

```
x(1498) = -3.50442
x(1499) = 1.89676
Time: 2.802936
```

Рисунок 6. Результат вычисления СЛАУ с помощью функции ParallelGaussMethod и время вычисления прямого хода метода Гаусса после добавления редьюсера

No Problems Detected
Intel Inspector detected no problems at this analysis scope. If this result is unexpected, try rerunning the target using an analysis type with a wider scope. Press F1 for more information.

Рисунок 7. Результат Inspector XE после добавления редьюсера

5. 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.

```
Solution:
x(0) = 1
x(1) = 2
x(2) = 2
x(3) = -0
Solution:
x(0) = 1
x(1) = 2
x(2) = 2
x(3) = -0
Serial time: 9e-07
Parallel time: 0.0005384
```

Рисунок 8. Результат вычисления СЛАУ для исходного массива (из задачи 1) и время вычисления

В результате первый вывод Solution относится к SerialGaussMethod, а второй к ParallelGaussMethod. Результаты вычисления схожи с результатами в 1 задании.

```
Serial time: 2.8101551  
Parallel time: 1.9767229
```

Рисунок 9. Время вычисления СЛАУ с размером матрицы MATRIX_SIZE

Использование параллелизма в вычисление СЛАУ с размером матрицы MATRIX_SIZE на 0,8334322 секунды быстрее последовательного вычисления.

Вывод:

В данной лабораторной работе параллелизм наконец показывает выигрыш больше чем на полсекунды. Также в данной работе вспомнили как пользоваться инструментами Amplifier XE и Inspector XE.