ДЗ к занятию 04.06.2021

Косенко Владимир, ИВТ-12М

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

2 Запустите первоначальную версию программы и получите решение для тестовой матрицы test_matrix, убедитесь в правильности приведенного алгоритма. Добавьте строки кода для измерения времени 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).

Запишем систему в виде матрицы

2	5	4	1	20
1	3	2	1	11
2	10	9	7	40
3	8	9	2	37

$$x_1 = \frac{37-8\cdot 2-9\cdot 2-2\cdot 0}{3} = \frac{3}{3} = 1$$

$$x_2 = \frac{46-9\cdot 2-17\cdot 0}{14} = \frac{28}{14} = 2$$

$$x_3 = \frac{-68-(-2)\cdot 0}{-34} = \frac{-68}{-34} = 2$$

$$x_4 = \frac{0}{24} = 0$$

Тестовая матрица:

```
Solution (elapsed time = 0.0000030 seconds):
x(0) = 1.000000
x(l) = 2.000000
x(2) = 2.000000
x(3) = -0.000000
```

Решение совпало.

Большая матрица:

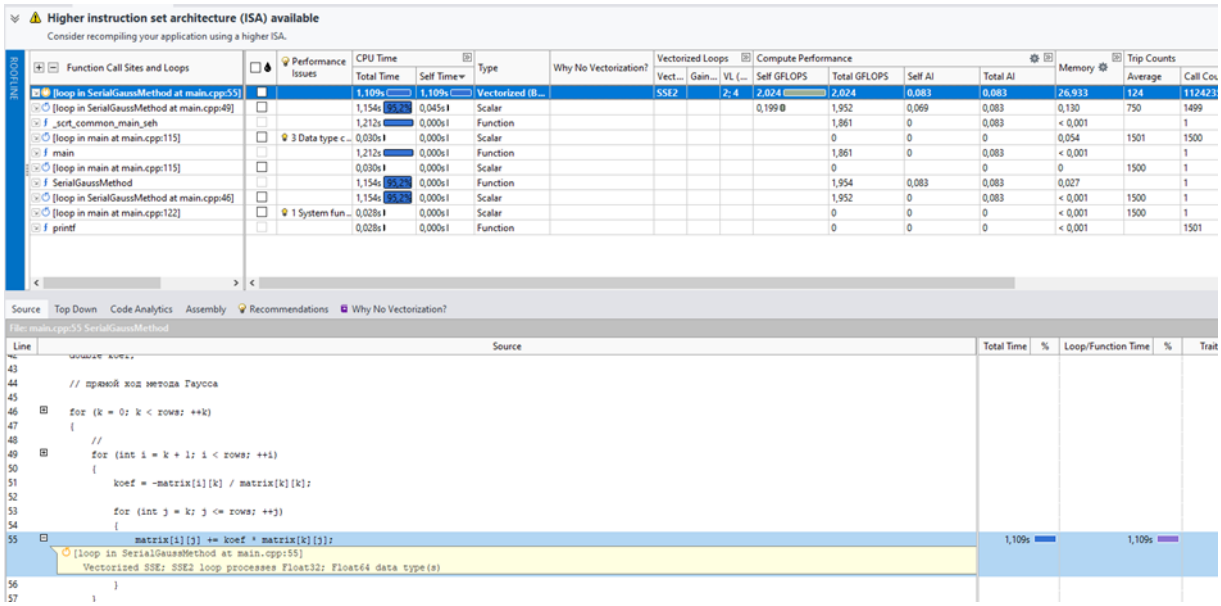
```
Solution (elapsed time = 1.09532930 seconds):
x(0) = -4.668498
x(0) = 6.145750
x(0) = -3.620851
x(0) = -4.999822
x(0) = 0.225535
x(0) = 3.454076
x(0) = 0.649761
```

ДЗ к занятию 04.06.2021

Косенко Владимир, ИВТ-12М

```
x(0) =  -2.246555
x(0) = -3.007229
x(0) = -4.978622
x(0) = -2.289153
x(0) = -4.635524
x(0) = 3.344822
x(0) = -0.256322
x(0) = -1.267626
x(0) = 2.872646
x(0) = -1.908741
x(0) = 0.336338
x(0) = -5.104676
x(0) = - 1.246713
x(0) = -1.239901
x(0) = - 0.280136
x(0) = - 0.910777
x(0) = - -1.593094
x(0) = - 1.304217
x(0) = - 4.496683
x(0) = -8.053281
x(0) = - 3.956933
x(0) = 2.041850
x(0) = - 1.661720
```

3 С помощью инструмента Intel Advisor определите наиболее часто используемые участки кода новой версии программы. Сохраните скриншот результатов анализа Intel Advisor. Создайте, на основе последовательной функции SerialGaussMethod(), новую функцию, реализующую параллельный метод Гаусса. Введите параллелизм в новую функцию, используя openmp. 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.



Инструмент указал на 55 строчку, которая и занимает больше всего процессорного времени, введем параллелизм этого в строке 48.

ДЗ к занятию 04.06.2021

Косенко Владимир, ИВТ-12М

```
73  chrono::duration<double> ParallelGaussMethod(double** matrix, const int rows, double* result)
74  {
75      int k;
76      double koef;
77
78      // прямой ход метода Гаусса
79      chrono::high_resolution_clock::time_point t1 = chrono::high_resolution_clock::now();
80      for (k = 0; k < rows; ++k) {
81  #pragma omp parallel for
82          for (int i = k + 1; i < rows; ++i) {
83              koef = -matrix[i][k] / matrix[k][k];
84              for (int j = k; j <= rows; ++j) {
85                  matrix[i][j] += koef * matrix[k][j];
86              }
87          }
88      }
89      chrono::high_resolution_clock::time_point t2 = chrono::high_resolution_clock::now();
90
91      // обратный ход метода Гаусса
92      result[rows - 1] = matrix[rows - 1][rows] / matrix[rows - 1][rows - 1];
93      for (k = rows - 2; k >= 0; --k) {
94          result[k] = matrix[k][rows];
95  #pragma omp parallel for
96          for (int j = k + 1; j < rows; ++j) {
97              result[k] -= matrix[k][j] * result[j];
98          }
99          result[k] /= matrix[k][k];
100     }
101
102     return chrono::duration<double>(t2 - t1);
103 }
```

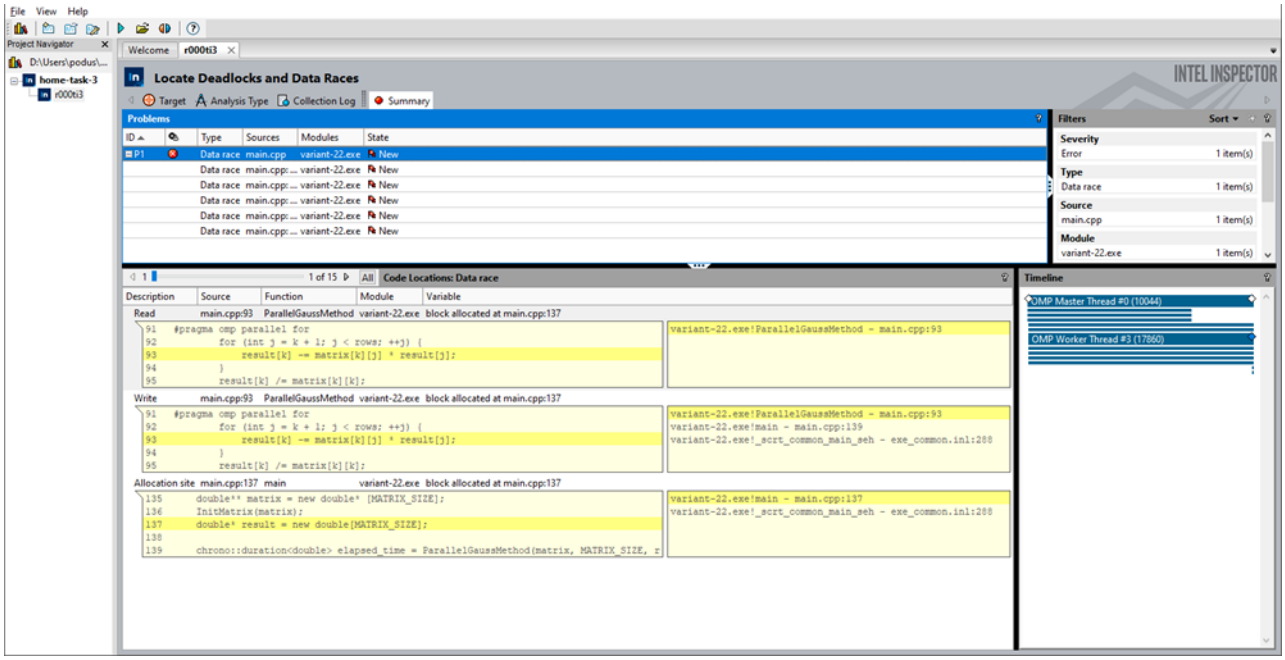
```
Parallel Big Solution (elapsed time = 0.64311060 seconds):
x(0) = -32042216154799278046172207246249002353324709464671763093091580860944277541407848841510999122070495821824.000000
x(1) = 10777915321747990449297480908880.000000
x(2) = -10207367209497277468291129091579867295228086211418008643004887904456540l6.000000
x(3) = 896584528331955008106403214019188436847836715630704481379655145176927556600718082368562223913946316800.000000
x(4) = -29937158534945213392099362280072037530176791243600035172332587939283834928581563242493l2.000000
x(5) = -82363106744533198165560104533549863628639315435235186769124108468748288.000000
x(6) = -35298905355593820111204618813548752957931112266641688620223685503941016384603546494368121949394126766O8.000000
x(7) = -66034584567006776192994488212982791072750403028084571300756776928042197701165542173081O.000000
x(8) = -88134806833978972026264065515074810753575059823001O.000000
x(9) = -23054110921370098304851767437476417524443798692513583821125392013022684989789538646984686135354316554l4.000000
x(10) = 19577040188810452618729945766010883259179867256536847893304358331498735908612912966411350712954828881l2.000000
x(11) = 232304569149895788900653203456.000000
x(12) = -12764077143858354987346098831282657702297781193014607510097951022358088190494633121299517616947l.000000
x(13) = 877861813030655258468398907235541357496990325345350767166642259978331822859845402113356460059659337728.000000
x(14) = -25906035746572757672008897665382484156181634803681567928264326356137533360890824770977792.000000
x(15) = 24295931042157718773866430464O.000000
x(16) = -25905782951245055887374991132638502167830547208951688159698297150079277424974513624992831704552964096.000000
x(17) = -39672626802133594041091750118545188811621049029377533418593805897838866323775060133810253578635313152.000000
x(18) = 10510366697933032574980042953883907118437298224262031237751829091006464188309954987075520201386413260l.000000
x(19) = 213520990941583717677500968631867379984564775676292156375961688468614335432170822964274535729779965952.000000
x(20) = 14874280530173657332865273350106670264266763123173663759816058428832211496203308859371347247104.000000
x(21) = 18769233401472091344864790633648566713976622025263115388353035448039909880822569702880791704371l.000000
x(22) = 536652162542388759932777054297123454326365795385344.000000
x(23) = -29593137362272029370651066119329156126951045547090181444381906162837094l.000000
x(24) = -63812413411350721497921246347580676428458475527399827355357700173585945562336370128369542798648541184.000000
x(25) = 73408014202279098476709820697507258451866451747077316650937102037928470041580994758759543734272.000000
x(26) = -47010514477329557395143408512.000000
x(27) = 22216985499098421845998069350l.000000
x(28) = -48210901657291369033935543245458570607021872915066922380402868168587382133018686776666962510789738496.000000
```

ДЗ к занятию 04.06.2021

Косенко Владимир, ИВТ-12М

4 Далее, используя Intel Inspector, 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 Intel Inspector: в случае обнаружения ошибок и после их устранения.

Результат анализа Инспектора после введения параллелизма:



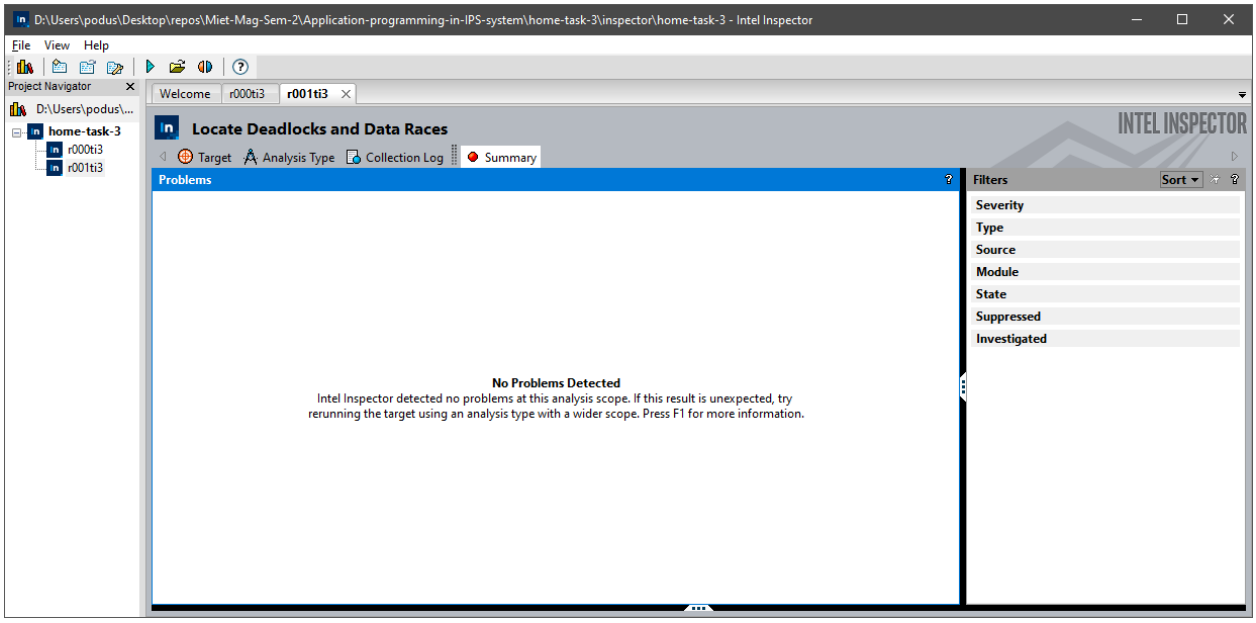
Инспектор указал на проблемы с записью и чтением общих данных (элементов матриц).

ДЗ к занятию 04.06.2021

Косенко Владимир, ИВТ-12М

Исправили проблему с данными, путем изменения кода и указания «reduction»

```
73  chrono::duration<double> ParallelGaussMethod(double** matrix, const int rows, double* result)
74  {
75      int k;
76      double koef;
77
78      // прямой ход метода Гаусса
79      chrono::high_resolution_clock::time_point t1 = chrono::high_resolution_clock::now();
80      for (k = 0; k < rows; ++k) {
81  #pragma omp parallel for
82          for (int i = k + 1; i < rows; ++i) {
83              koef = -matrix[i][k] / matrix[k][k];
84              double* matrix_i = matrix[i], *matrix_k = matrix[k];
85              for (int j = k; j <= rows; ++j) {
86                  matrix_i[j] += koef * matrix_k[j];
87              }
88          }
89      }
90      chrono::high_resolution_clock::time_point t2 = chrono::high_resolution_clock::now();
91
92      // обратный ход метода Гаусса
93      result[rows - 1] = matrix[rows - 1][rows] / matrix[rows - 1][rows - 1];
94      for (k = rows - 2; k >= 0; --k) {
95          //result[k] = matrix[k][rows];
96          double temp = 0;
97  #pragma omp parallel for reduction(+:temp)
98          for (int j = k + 1; j < rows; ++j) {
99              //result[k] -= matrix[k][j] * result[j];
100             temp += matrix[k][j] * result[j];
101         }
102         result[k] = matrix[k][rows] - temp;
103         result[k] /= matrix[k][k];
104     }
```



ДЗ к занятию 04.06.2021

Косенко Владимир, ИВТ-12М

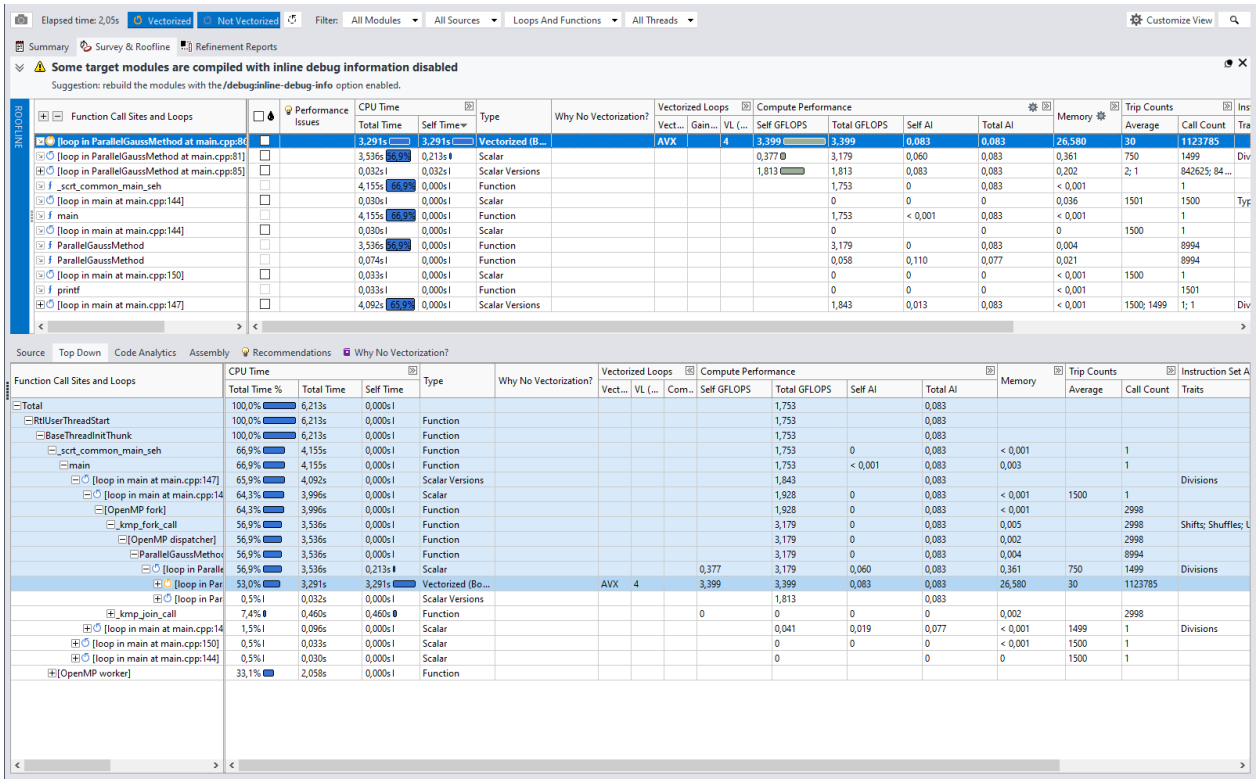
5 Убедитесь на примере тестовой матрицы test_matrix в том, что функция, реализующая параллельный метод Гаусса работает правильно. Сравните время выполнения прямого хода метода Гаусса для последовательной и параллельной реализации при решении матрицы, имеющей количество строк MATRIX_SIZE, заполняющейся 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.

Убеждаемся в правильной работе распараллеленной программы на тестовой матрице:

```
Solution (elapsed time = 0.0000030 seconds):
x(0) = 1.000000
x(l) = 2.000000
x(2) = 2.000000
x(3) = -0.000000
```

Решение верное.

Проанализируем работу модифицированной программы на большой матрице:



ДЗ к занятию 04.06.2021

Косенко Владимир, ИВТ-12М

```
Parallel Big Solution (elapsed time = 0.72843940 seconds):
x(0) = 16.803467
x(1) = -19.649955
x(2) = -17.103566
x(3) = 12.811761
x(4) = -2.656953
x(5) = -5.288017
x(6) = 15.280094
x(7) = -18.686445
x(8) = -9.082980
x(9) = 9.561928
x(10) = 9.477927
x(11) = 8.192563
x(12) = 12.685736
x(13) = 8.626219
x(14) = 1.307444
x(15) = -10.396956
x(16) = 1.521483
x(17) = -2.842237
x(18) = -24.379276
x(19) = -3.341244
x(20) = -21.083549
x(21) = -30.541293
x(22) = 20.446882
x(23) = -14.054809
x(24) = 31.002040
x(25) = -33.500527
x(26) = 9.297572
x(27) = -2.720279
x(28) = -11.094550
```

Используя Intel Adviser, убеждаемся, что программа выполнялась параллельно.

Результат вычисления совпал с последовательной программой.

Время работы параллельной программы: 0.72843940 секунд

Время работы последовательной программы: 1.082821130 секунд

Примерно в 1.5 раза быстрее стали вычислять решение поставленной задачи.

6 Результатом работы должна быть ссылка на Ваш профиль GitHub где представлен код программы с введенными изменениями согласно заданиям. Все скриншоты (отражающие работу с Intel Advisor и Intel Inspector, а также полученное значение ускорения) представьте в .pdf файле и загрузите в систему в качестве отчета.

Ссылка на папку в репозитории

https://github.com/Concor/IPS_labs/tree/master/IPS_lab4

Все проекты в одном решении