Saint Petersburg National Research University of Information Technologies, Mechanics and Optics (ITMO University)

REPORT

about laboratory works

Assignment 0

Assignment 1. OpenMP. Finding the maximum value of a vector

Assignment 2. OpenMP. Matrix multiplication

Student Pogrebnoy D.A.   j4132c

Saint-Petersburg, 2021

ASSIGNMENT 0.

Task

Write a program for counting words in a line. Any sequence if cheracters without separators is considered as a word. Separators are spaces, tabs, newlines. The input string is passed to the program through the terminal as the argv[1] parameter. The program should display the number of words on the screen.

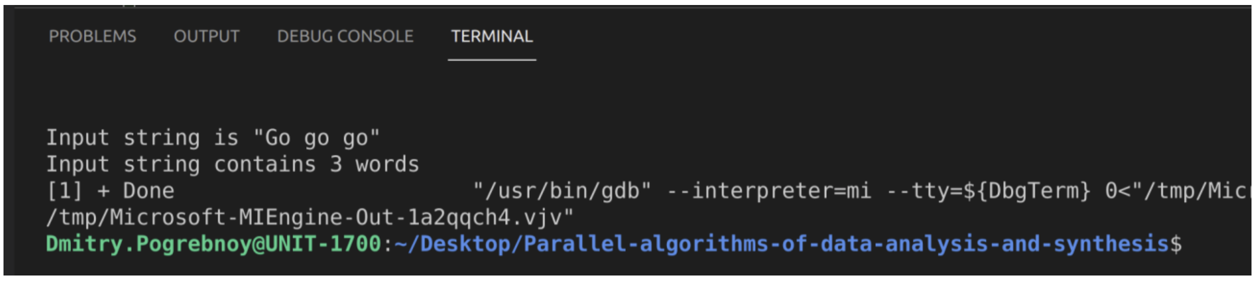
Implementation

The code was implemented using C++ and compiled with g++ compiler (version 9.3.0, Ubuntu 20.04). The processor used during measurements was Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz with 6 physical and 12 virtual cores.

Source code and data gathered are available on https://github.com/DmitryPogrebnoy/Parallel-algorithms-of-data-analysis-and-synthesis/blob/master/OmpTasks/Task0/Task0.cpp.

The program accepts a string as input. The program looks for the first separating character in the string in a loop, and if there is one, it updates the pointers of the current positions in the string and increases the word counter. Upon exiting the loop, the program prints the found number of words in the input string.

For example, let's input string is "Go go go" and program shows result as follows.



```
Input string is "Go go go"
Input string contains 3 words
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Mic
/tmp/Microsoft-MIEngine-Out-1a2qqch4.vjv"
Dmitry.Pogrebnoy@UNIT-1700:~/Desktop/Parallel-algorithms-of-data-analysis-and-synthesis$
```

ASSIGMENTS 1. OPENMP. FINDING THE MAXIMUM VALUE OF A VECTOR

Task

Write a parallel OpenMP program that finds the maximum value of a vector (one-dimensional array). Each thread should only store its maximum value; concurrent access to a shared variable that stores the maximum value is not allowed. Study the dependence of the runtime on the number of threads used (from 1 to 10) for a vector that contains at least 1,000,000 elements (the more, the better). Check the correctness of the program on 10 elements. The program should display on the screen: the number of threads, the execution time. Transfer the size of the vector through the argv [1] parameter.

Implementation

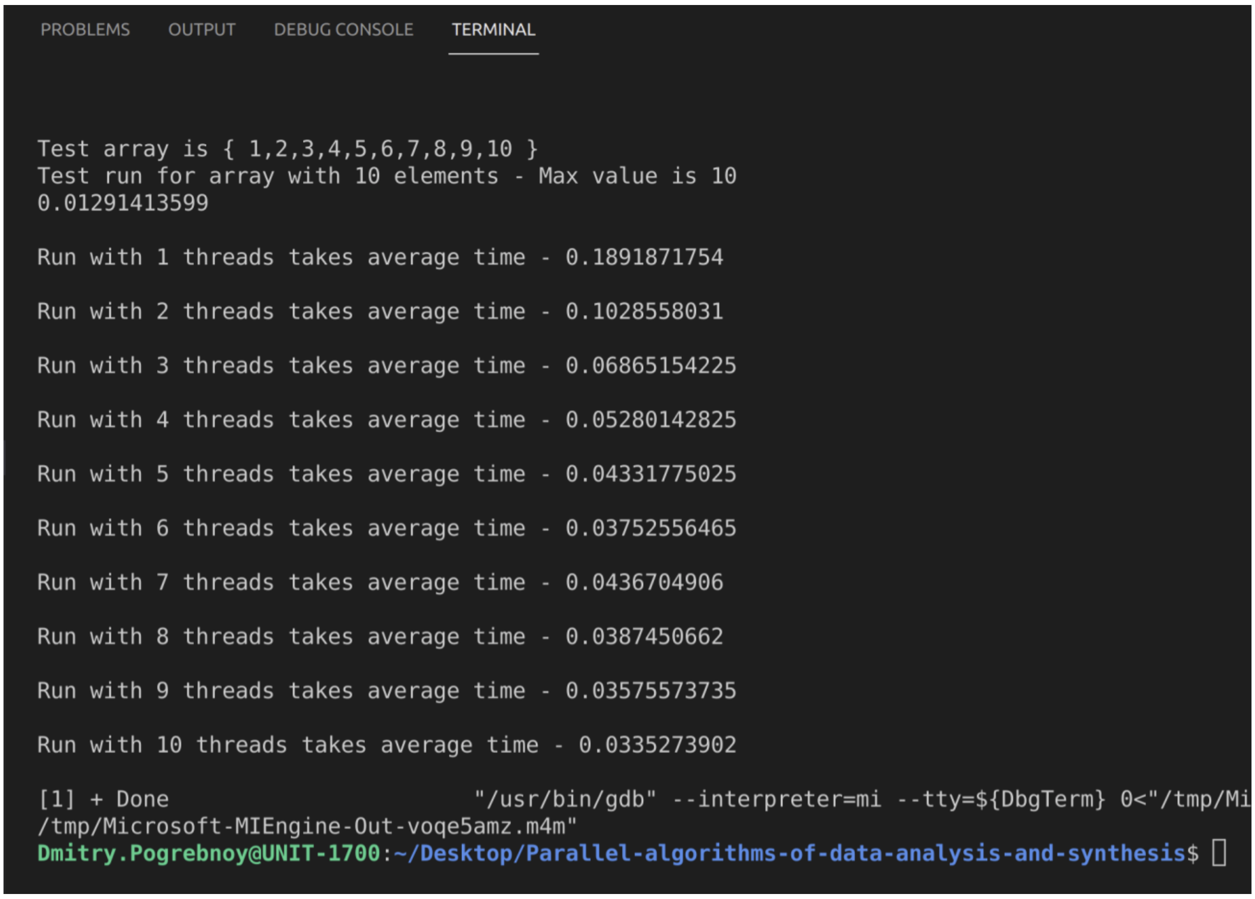
The code was implemented using C++ and compiled with g++ compiler (version 9.3.0, Ubuntu 20.04). The processor used during measurements was Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz with 6 physical and 12 virtual cores.

Source code and data gathered are available on https://github.com/DmitryPogrebnoy/Parallel-algorithms-of-data-analysis-and-synthesis/blob/master/OmpTasks/Task1/Task1.cpp.

The program accepts N vector length. Next, the maximum value in a text array of 12 threads is calculated to check the correctness of the algorithm. After that, a vector with a length of 100,000,000 is randomly generated and the maximum value is searched in it. This procedure is performed with a different number of threads for the test. For each number of threads, 20 identical runs are performed and the time is averaged. To calculate the maximum element, parallelization of the for loop using OpenMP and the reduction statement are used.

reduction – (reduction clause) the variable has a local copy in each thread, but the values of the local copies will be summarized (reduced) into a global shared variable. This is very useful if a particular operation (specified in operator for this particular clause) on a variable runs iteratively, so that its value at a particular iteration depends on its value at a prior iteration. The steps that lead up to the operational increment are parallelized, but the threads updates the global variable in a thread safe manner.

The results of the launches are presented below. They clearly show a gradual increase in performance as a result of a larger number of threads.



```
Test array is { 1,2,3,4,5,6,7,8,9,10 }
Test run for array with 10 elements - Max value is 10
0.01291413599

Run with 1 threads takes average time - 0.1891871754

Run with 2 threads takes average time - 0.1028558031

Run with 3 threads takes average time - 0.06865154225

Run with 4 threads takes average time - 0.05280142825

Run with 5 threads takes average time - 0.04331775025

Run with 6 threads takes average time - 0.03752556465

Run with 7 threads takes average time - 0.0436704906

Run with 8 threads takes average time - 0.0387450662

Run with 9 threads takes average time - 0.03575573735

Run with 10 threads takes average time - 0.0335273902

[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Mi
/tmp/Microsoft-MIEngine-Out-voqe5amz.m4m"
Dmitry.Pogrebnoy@UNIT-1700:~/Desktop/Parallel-algorithms-of-data-analysis-and-synthesis$ 
```

ASSIGNMENT 2. OPENMP. MATRIX MULTIPLICATION

Task

Write a program for multiplying two square matrices using OpenMP. Examine the performance of different modifications of the algorithm (different loop order), depending on the number of threads used for matrices of at least 800x800. Check the correctness of the multiplication on 5x5 matrices. Calculate the efficiency by the formula t1 / t and display it, where t1 is the multiplition time on only one stream, t is the multiplication time on n streams (the number of streams is taken from 1 to 10). The program should output number of threads, multiplication time and efficiency. Transfer the size of matrices through the argv[] parameter.

Implementation

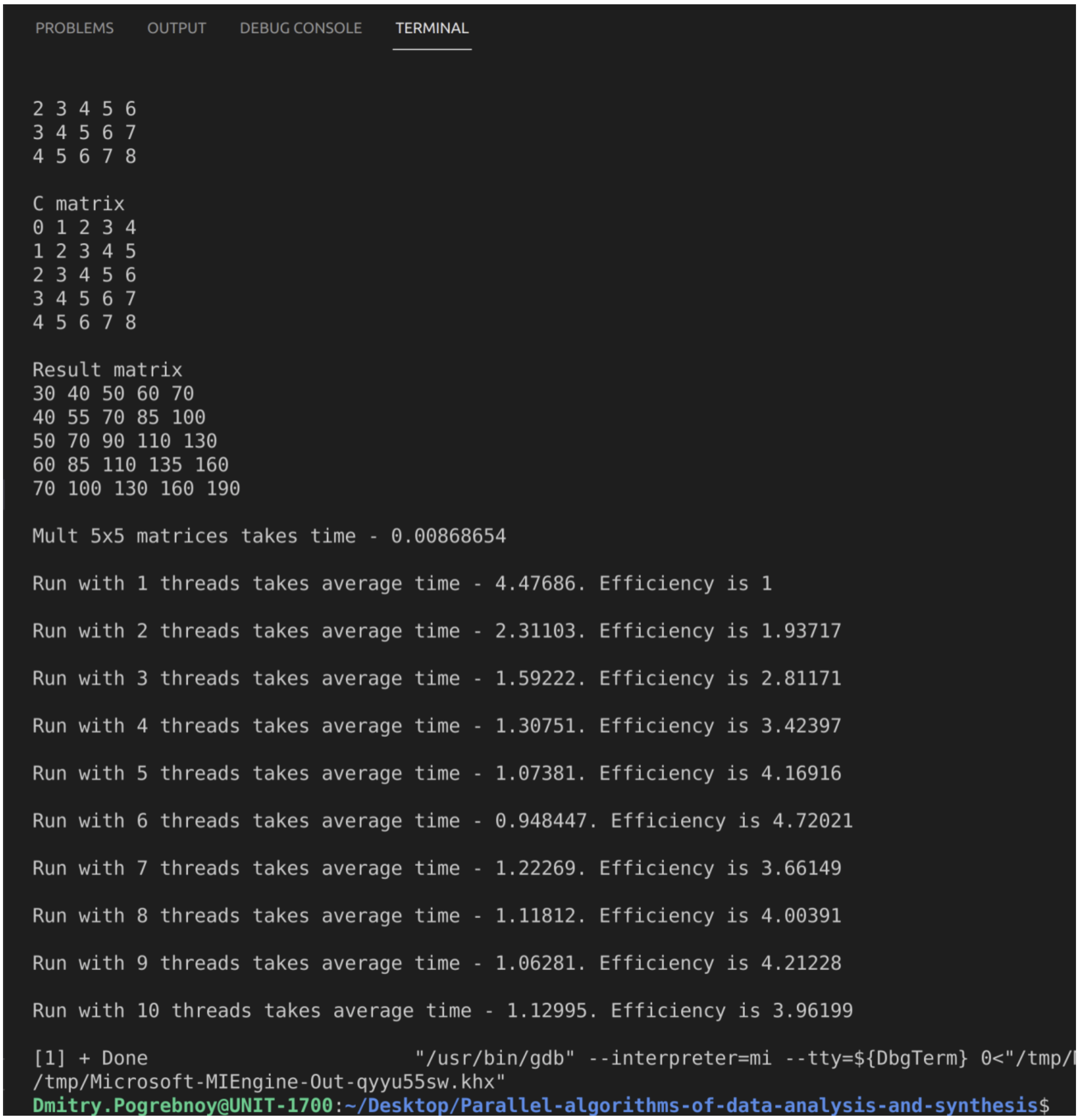
The code was implemented using C++ and compiled with g++ compiler (version 9.3.0, Ubuntu 20.04). The processor used during measurements was Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz with 6 physical and 12 virtual cores.

Source code and data gathered are available on https://github.com/DmitryPogrebnoy/Parallel-algorithms-of-data-analysis-and-synthesis/blob/master/OmpTasks/Task2/Task2.cpp.

The program accepts M, N, P sizes of input matrices. Next, the product of two matrices in a test array of 12 threads is calculated to check the correctness of the algorithm. The result is displayed on the screen. After that, three matrices of the appropriate size are randomly generated (if the parameters are not passed, then 1000x1000 matrices will be generated). The algorithm for multiplying two matrices is paralleled using OpenMP directives. The Shared directive is used to grant access to all three matrices between threads. The Private directive makes counters private per threads. Scheduled provides a cyclic order of execution of threads.

Also, several runs are performed for each number of threads, and the time is averaged. In addition, an increase in speed is calculated compared to the single-threaded version of the program.

The result is presented below. They clearly show a gradual increase in performance as a result of a larger number of threads.



```
2 3 4 5 6
3 4 5 6 7
4 5 6 7 8

C matrix
0 1 2 3 4
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
4 5 6 7 8

Result matrix
30 40 50 60 70
40 55 70 85 100
50 70 90 110 130
60 85 110 135 160
70 100 130 160 190

Mult 5x5 matrices takes time - 0.00868654

Run with 1 threads takes average time - 4.47686. Efficiency is 1

Run with 2 threads takes average time - 2.31103. Efficiency is 1.93717

Run with 3 threads takes average time - 1.59222. Efficiency is 2.81171

Run with 4 threads takes average time - 1.30751. Efficiency is 3.42397

Run with 5 threads takes average time - 1.07381. Efficiency is 4.16916

Run with 6 threads takes average time - 0.948447. Efficiency is 4.72021

Run with 7 threads takes average time - 1.22269. Efficiency is 3.66149

Run with 8 threads takes average time - 1.11812. Efficiency is 4.00391

Run with 9 threads takes average time - 1.06281. Efficiency is 4.21228

Run with 10 threads takes average time - 1.12995. Efficiency is 3.96199

[1] + Done                         "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/
/tmp/Microsoft-MIEngine-Out-qyyu55sw.khx"
Dmitry.Pogrebnoy@UNIT-1700:~/Desktop/Parallel-algorithms-of-data-analysis-and-synthesis$
```