FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION OF HIGHER EDUCATION

ITMO UNIVERSITY

Report

on the practical task No. 0,1,2

Performed by

*Alexander Yamoldin*

*J4134c*

St. Petersburg 2021

# Goal

Understand basic C++ syntax, use OpenMP library to find maximum number in a vector and experimental study of the time complexity of matrix multiplications.

# Formulation of the problem

In the Assignment 0 need to write a program for counting words in a line. Any sequence of characters without separators is considered 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.

In the Assignment 1 need to 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.

*In the assignment 2 need to w*rite 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 multiplication 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.

# Results

The code of the Assignment 0 can be found in

https://github.com/AAYamoldin/TrainingPrograms/blob/master/institute\_c\_programs/ITMO\_Parallel\_Algorithm/Task\_0/A0.cpp

The result of the program in the picture below:



The code of the Assignment 1 can be found in

https://github.com/AAYamoldin/TrainingPrograms/blob/master/institute\_c\_programs/ITMO\_Parallel\_Algorithm/Task\_1/A1.cpp

The result of the program is shown below:

root@DESKTOP-6HRJV3D:~/TrainingPrograms/institute\_c\_programs/ITMO\_Parallel\_Algorithm/Task\_1# ./A1.o "100000"

Num threads: 1

Max value: 2147469841

Time execution: 0.000389

Num threads: 2

Max value: 2147469841

Time execution: 0.000256

Num threads: 3

Max value: 2147469841

Time execution: 0.000231

Num threads: 4

Max value: 2147469841

Time execution: 0.000278

Num threads: 5

Max value: 2147469841

Time execution: 0.000257

Num threads: 6

Max value: 2147469841

Time execution: 0.000299

Num threads: 7

Max value: 2147469841

Time execution: 0.000296

Num threads: 8

Max value: 2147469841

Time execution: 0.000292

Num threads: 9

Max value: 2147469841

Time execution: 0.000301

Num threads: 10

Max value: 2147469841

Time execution: 0.000350

Num threads: 11

Max value: 2147469841

Time execution: 0.001641

Num threads: 12

Max value: 2147469841

Time execution: 0.002920

The code of the Assignment 2 can be found in

https://github.com/AAYamoldin/TrainingPrograms/blob/master/institute\_c\_programs/ITMO\_Parallel\_Algorithm/Task\_2/A3.cpp

The result of program is shown below:

root@DESKTOP-6HRJV3D:~/TrainingPrograms/institute\_c\_programs/ITMO\_Parallel\_Algorithm/Task\_2# ./A3.o "1000"

i\*j\*k multiplication

Threads 1 Time 6.67185 seconds, efficiency: 1

Threads 2 Time 3.47281 seconds, efficiency: 1.92116

Threads 3 Time 2.43524 seconds, efficiency: 2.73971

Threads 4 Time 1.97634 seconds, efficiency: 3.37586

Threads 5 Time 1.67365 seconds, efficiency: 3.9864

Threads 6 Time 1.56071 seconds, efficiency: 4.27487

Threads 7 Time 1.51792 seconds, efficiency: 4.39539

Threads 8 Time 1.3496 seconds, efficiency: 4.94359

Threads 9 Time 1.25269 seconds, efficiency: 5.32602

Threads 10 Time 1.15447 seconds, efficiency: 5.77914

Threads 11 Time 1.29017 seconds, efficiency: 5.1713

Threads 12 Time 1.36754 seconds, efficiency: 4.87872

j\*k\*i multiplication

Threads 1 Time 12.8177 seconds, efficiency: 1

Threads 2 Time 7.19332 seconds, efficiency: 1.78189

Threads 3 Time 4.7863 seconds, efficiency: 2.67799

Threads 4 Time 3.94979 seconds, efficiency: 3.24515

Threads 5 Time 3.50486 seconds, efficiency: 3.65712

Threads 6 Time 3.20112 seconds, efficiency: 4.00412

Threads 7 Time 2.83022 seconds, efficiency: 4.52886

Threads 8 Time 2.53678 seconds, efficiency: 5.05273

Threads 9 Time 2.29768 seconds, efficiency: 5.57853

Threads 10 Time 2.19806 seconds, efficiency: 5.83136

Threads 11 Time 2.15427 seconds, efficiency: 5.9499

Threads 12 Time 2.28931 seconds, efficiency: 5.59893

i\*k\*j multiplication

Threads 1 Time 3.30787 seconds, efficiency: 1

Threads 2 Time 1.81997 seconds, efficiency: 1.81754

Threads 3 Time 1.32672 seconds, efficiency: 2.49327

Threads 4 Time 1.09795 seconds, efficiency: 3.01278

Threads 5 Time 0.901707 seconds, efficiency: 3.66846

Threads 6 Time 0.843011 seconds, efficiency: 3.92388

Threads 7 Time 0.781548 seconds, efficiency: 4.23246

Threads 8 Time 0.679672 seconds, efficiency: 4.86687

Threads 9 Time 0.641298 seconds, efficiency: 5.15809

Threads 10 Time 0.589881 seconds, efficiency: 5.6077

Threads 11 Time 0.562806 seconds, efficiency: 5.87746

Threads 12 Time 0.571403 seconds, efficiency: 5.78903

# Conclusions

From the obtained in the Assignment 0 we can see a right number of words in a string.

In the Assignment 1 we obtain the best result with using 3 threads (evaluation time: 0.000231s.), the worst result is with using 12 thread (evaluation time: 0.002920s.) so the difference between the best and the worst solutions are 12.6 times. Using parallel paradigm give a gain of a performance by 1.7 times against a single thread evaluation.

In the Assignment 2 we obtain the best result in i\*j\*k multiplication with using 10 thread (evaluation time: 1.15447 s.) the worst result with using 1 thread (evaluation time: 6.67185s.) parallel paradigm in this case give a gain of performance by 5.77 times, in j\*k\*i multiplication with using 11 thread (evaluation time: 2.15427 s.) the worst result with using 1 thread (evaluation time: 12.8177 s.) parallel paradigm in this case give a gain of performance by 5.95 times, , in i\*k\*j multiplication with using 11 thread (evaluation time: 0.562806 s.) the worst result with using 1 thread (evaluation time: 3.30787s..) parallel paradigm in this case give a gain of performance by 5.88 times. In comparing the best solutions we obtain that the fastest evaluation is by i\*k\*j multiplication. This algorithm faster than i\*j\*k by 2 times, and than j\*k\*I by 3.83 times. We can see that not always the more threads the better. Number of threads depend on task and are individually for the each task and processors architecture.