FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION
OF HIGHER EDUCATION
ITMO UNIVERSITY

Parallel algorithms for the analysis and synthesis of data on the assignment No.0, 1, 2

Performed by
Ivan Dubinin
J4132c

St. Petersburg
2021

Assignment 0

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.

Listing of the program

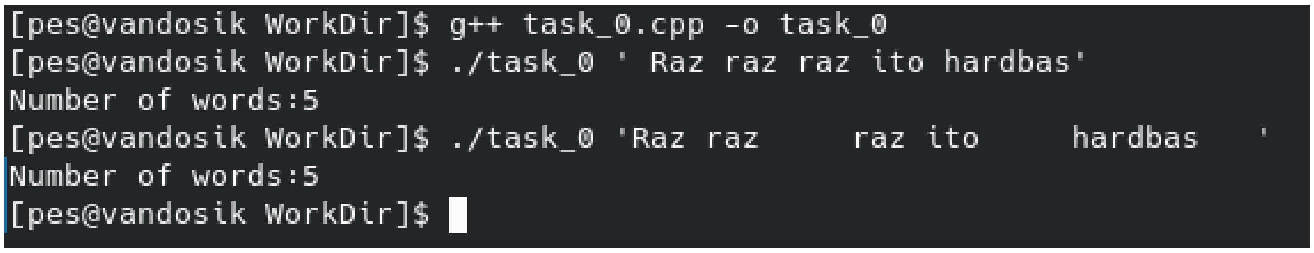
See it in my repo

Description

At the beginning If the nubmer or argements passed to programm is bigger than 1 (first argument is the string with prog file path, used for launch), we continue. Otherwise we return with code 1.

Than I iterate over got string per each symbol, where I save the previos symbol and compare it with new one. If the prev symbol is not separator and the new one is – than we reached the end of some word. So we increment the counter of words.

Example of launch parameters and output



```
[pes@vandosik WorkDir]$ g++ task_0.cpp -o task_0
[pes@vandosik WorkDir]$ ./task_0 ' Raz raz raz ito hardbas'
Number of words:5
[pes@vandosik WorkDir]$ ./task_0 'Raz raz     raz ito     hardbas   '
Number of words:5
[pes@vandosik WorkDir]$ 
```

Assignment 1

Write a parallel OpenMP program that finds the maximum value of a vector (onedimensional 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.

Listing of the program

See it in my repo

Description

At the beginning,we get the size of the vector from argv[1]. Then we create a std::vector and fill it with random doubles from 0 to 1.

Later we iterate over different numbers of threads and for each one calculate the execution time of finding maximum value of the vector. As one can see on figure below the amount of time need for calculation is decresing over these 10 numbers of freads(for 10 threads it is faster then for 1) with 10 mln elems. This programm was tested with AMD Ryzen 5 with 16 Logical cores.

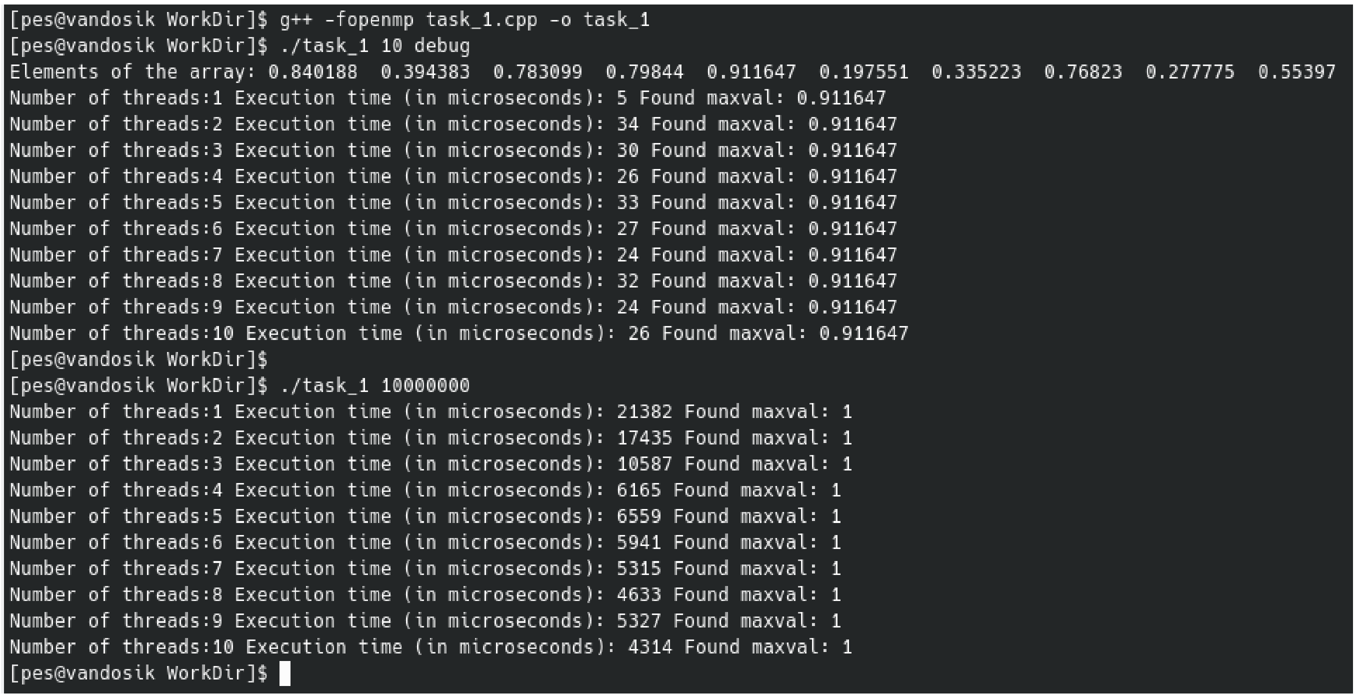
The first launch on the picture shows that the programm is workable.

For introducing parralelism we use:

#pragma omp parallel for num_threads(thread_num) reduction (max:max_val)

Reduction creates private variable for each thread and then assembles them in max option. By num_threads() we set the number of threads to execute with.

Example of launch parameters and output



```
[pes@vandosik WorkDir]$ g++ -fopenmp task_1.cpp -o task_1
[pes@vandosik WorkDir]$ ./task_1 10 debug
Elements of the array: 0.840188  0.394383  0.783099  0.79844  0.911647  0.197551  0.335223  0.76823  0.277775  0.55397
Number of threads:1 Execution time (in microseconds): 5 Found maxval: 0.911647
Number of threads:2 Execution time (in microseconds): 34 Found maxval: 0.911647
Number of threads:3 Execution time (in microseconds): 30 Found maxval: 0.911647
Number of threads:4 Execution time (in microseconds): 26 Found maxval: 0.911647
Number of threads:5 Execution time (in microseconds): 33 Found maxval: 0.911647
Number of threads:6 Execution time (in microseconds): 27 Found maxval: 0.911647
Number of threads:7 Execution time (in microseconds): 24 Found maxval: 0.911647
Number of threads:8 Execution time (in microseconds): 32 Found maxval: 0.911647
Number of threads:9 Execution time (in microseconds): 24 Found maxval: 0.911647
Number of threads:10 Execution time (in microseconds): 26 Found maxval: 0.911647
[pes@vandosik WorkDir]$
[pes@vandosik WorkDir]$ ./task_1 10000000
Number of threads:1 Execution time (in microseconds): 21382 Found maxval: 1
Number of threads:2 Execution time (in microseconds): 17435 Found maxval: 1
Number of threads:3 Execution time (in microseconds): 10587 Found maxval: 1
Number of threads:4 Execution time (in microseconds): 6165 Found maxval: 1
Number of threads:5 Execution time (in microseconds): 6559 Found maxval: 1
Number of threads:6 Execution time (in microseconds): 5941 Found maxval: 1
Number of threads:7 Execution time (in microseconds): 5315 Found maxval: 1
Number of threads:8 Execution time (in microseconds): 4633 Found maxval: 1
Number of threads:9 Execution time (in microseconds): 5327 Found maxval: 1
Number of threads:10 Execution time (in microseconds): 4314 Found maxval: 1
[pes@vandosik WorkDir]$
```

Assignment 2

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 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.

Listing of the program

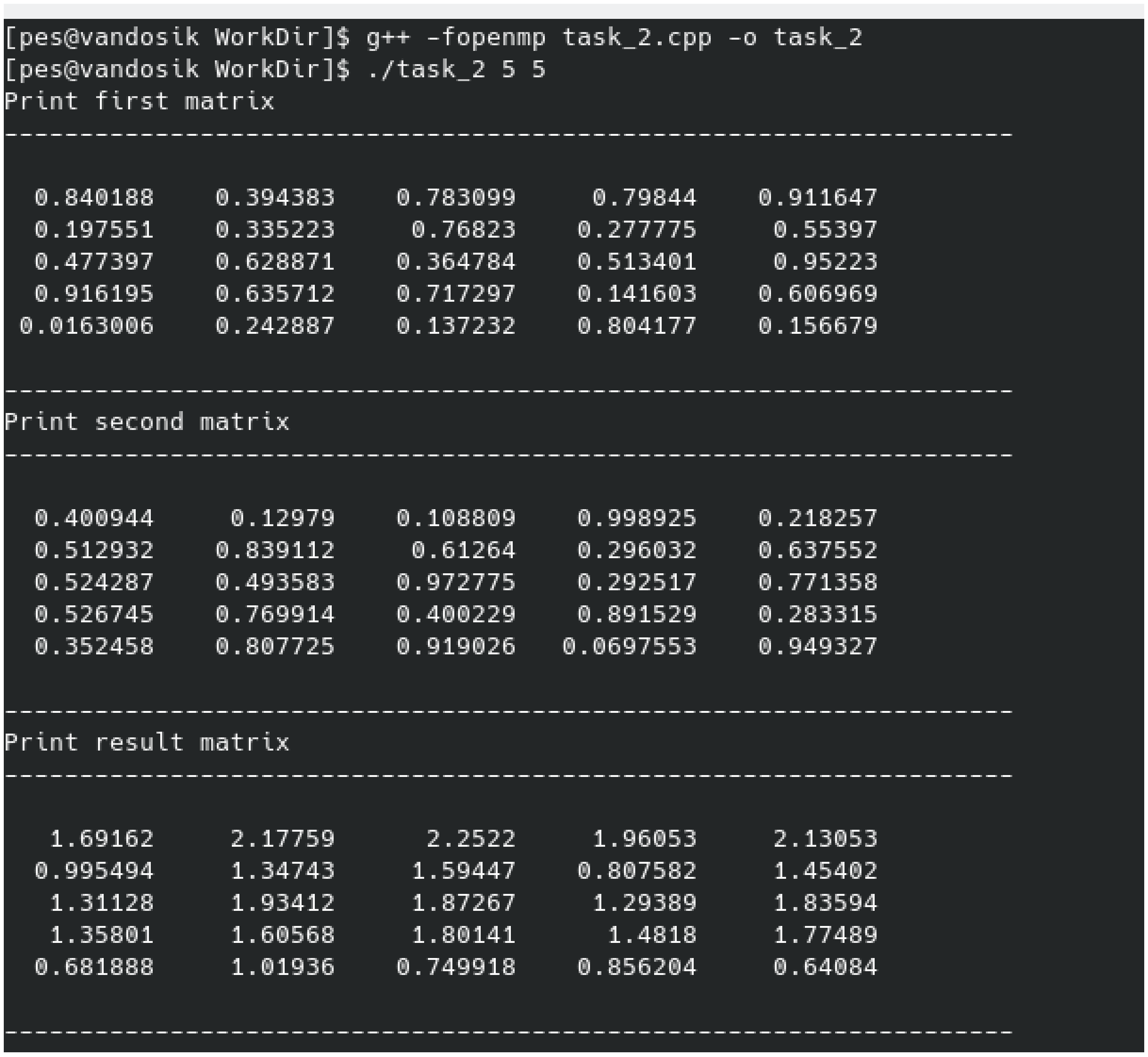
See it in my repo

Description

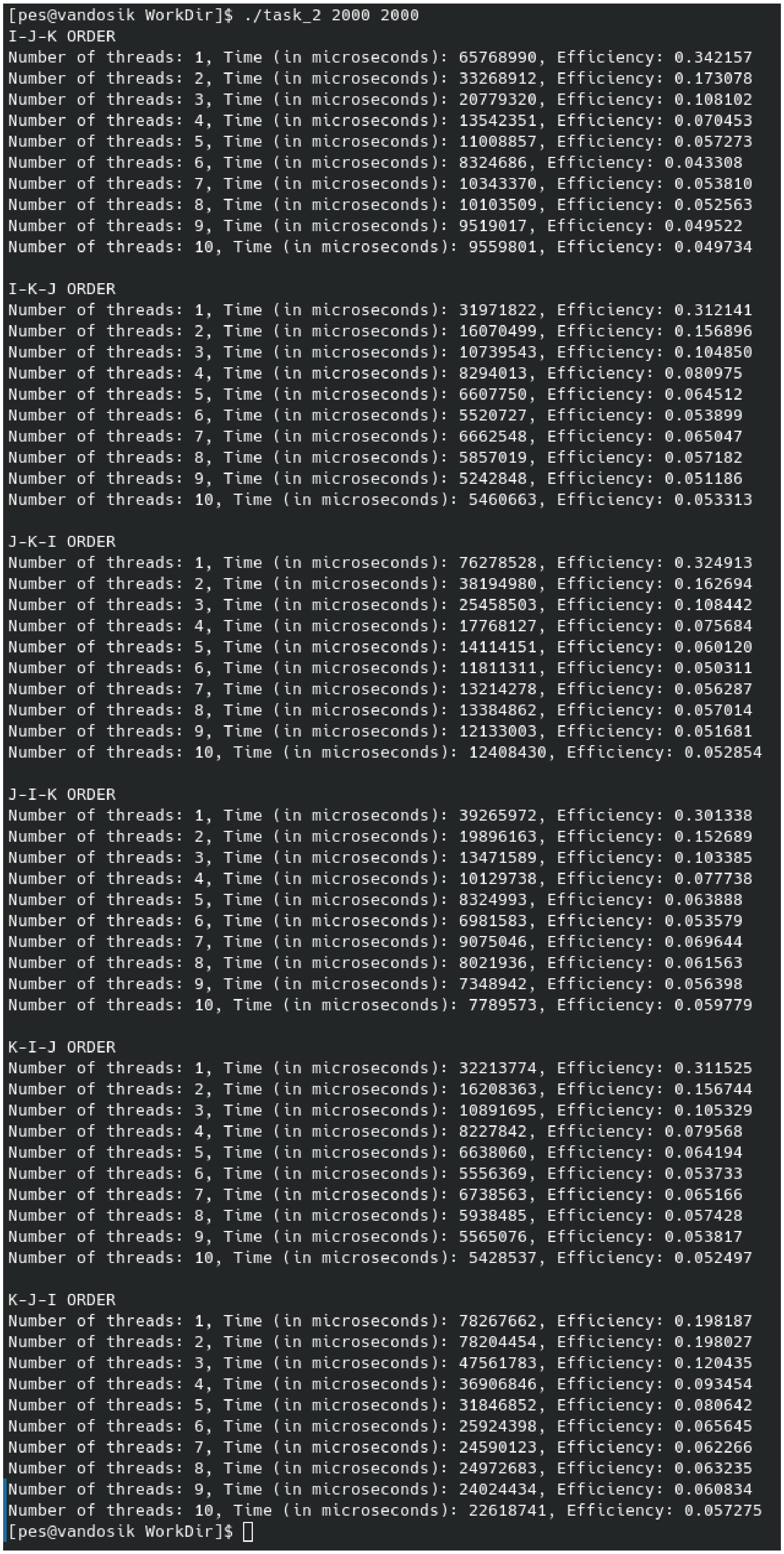
On figure 1 one can see the check of the workability of programm with 5*5 matrixes. In figure 2 the resilts of efficiency calculations is shown for each ordet for 2000*2000 matrixes.

Considering results the best performance is shown by IKJ order. Epy KIJ order shows also wery good results, but a bit clower, than IKJ

Example of launch parameters and output



```
[pes@vandosik WorkDir]$ g++ -fopenmp task_2.cpp -o task_2
[pes@vandosik WorkDir]$ ./task_2 5 5
Print first matrix
-------------------------------------------------------------------

  0.840188    0.394383    0.783099     0.79844    0.911647
  0.197551    0.335223     0.76823    0.277775     0.55397
  0.477397    0.628871    0.364784    0.513401     0.95223
  0.916195    0.635712    0.717297    0.141603    0.606969
 0.0163006    0.242887    0.137232    0.804177    0.156679

-------------------------------------------------------------------
Print second matrix
-------------------------------------------------------------------

  0.400944     0.12979    0.108809    0.998925    0.218257
  0.512932    0.839112     0.61264    0.296032    0.637552
  0.524287    0.493583    0.972775    0.292517    0.771358
  0.526745    0.769914    0.400229    0.891529    0.283315
  0.352458    0.807725    0.919026   0.0697553    0.949327

-------------------------------------------------------------------
Print result matrix
-------------------------------------------------------------------

   1.69162     2.17759      2.2522     1.96053     2.13053
  0.995494     1.34743     1.59447    0.807582     1.45402
   1.31128     1.93412     1.87267     1.29389     1.83594
   1.35801     1.60568     1.80141      1.4818     1.77489
  0.681888     1.01936    0.749918    0.856204     0.64084

-------------------------------------------------------------------
```



```
[pes@vandosik WorkDir]$ ./task_2 2000 2000
I-J-K ORDER
Number of threads: 1, Time (in microseconds): 65768990, Efficiency: 0.342157
Number of threads: 2, Time (in microseconds): 33268912, Efficiency: 0.173078
Number of threads: 3, Time (in microseconds): 20779320, Efficiency: 0.108102
Number of threads: 4, Time (in microseconds): 13542351, Efficiency: 0.070453
Number of threads: 5, Time (in microseconds): 11008857, Efficiency: 0.057273
Number of threads: 6, Time (in microseconds): 8324686, Efficiency: 0.043308
Number of threads: 7, Time (in microseconds): 10343370, Efficiency: 0.053810
Number of threads: 8, Time (in microseconds): 10103509, Efficiency: 0.052563
Number of threads: 9, Time (in microseconds): 9519017, Efficiency: 0.049522
Number of threads: 10, Time (in microseconds): 9559801, Efficiency: 0.049734

I-K-J ORDER
Number of threads: 1, Time (in microseconds): 31971822, Efficiency: 0.312141
Number of threads: 2, Time (in microseconds): 16070499, Efficiency: 0.156896
Number of threads: 3, Time (in microseconds): 10739543, Efficiency: 0.104850
Number of threads: 4, Time (in microseconds): 8294013, Efficiency: 0.080975
Number of threads: 5, Time (in microseconds): 6607750, Efficiency: 0.064512
Number of threads: 6, Time (in microseconds): 5520727, Efficiency: 0.053899
Number of threads: 7, Time (in microseconds): 6662548, Efficiency: 0.065047
Number of threads: 8, Time (in microseconds): 5857019, Efficiency: 0.057182
Number of threads: 9, Time (in microseconds): 5242848, Efficiency: 0.051186
Number of threads: 10, Time (in microseconds): 5460663, Efficiency: 0.053313

J-K-I ORDER
Number of threads: 1, Time (in microseconds): 76278528, Efficiency: 0.324913
Number of threads: 2, Time (in microseconds): 38194980, Efficiency: 0.162694
Number of threads: 3, Time (in microseconds): 25458503, Efficiency: 0.108442
Number of threads: 4, Time (in microseconds): 17768127, Efficiency: 0.075684
Number of threads: 5, Time (in microseconds): 14114151, Efficiency: 0.060120
Number of threads: 6, Time (in microseconds): 11811311, Efficiency: 0.050311
Number of threads: 7, Time (in microseconds): 13214278, Efficiency: 0.056287
Number of threads: 8, Time (in microseconds): 13384862, Efficiency: 0.057014
Number of threads: 9, Time (in microseconds): 12133003, Efficiency: 0.051681
Number of threads: 10, Time (in microseconds): 12408430, Efficiency: 0.052854

J-I-K ORDER
Number of threads: 1, Time (in microseconds): 39265972, Efficiency: 0.301338
Number of threads: 2, Time (in microseconds): 19896163, Efficiency: 0.152689
Number of threads: 3, Time (in microseconds): 13471589, Efficiency: 0.103385
Number of threads: 4, Time (in microseconds): 10129738, Efficiency: 0.077738
Number of threads: 5, Time (in microseconds): 8324993, Efficiency: 0.063888
Number of threads: 6, Time (in microseconds): 6981583, Efficiency: 0.053579
Number of threads: 7, Time (in microseconds): 9075046, Efficiency: 0.069644
Number of threads: 8, Time (in microseconds): 8021936, Efficiency: 0.061563
Number of threads: 9, Time (in microseconds): 7348942, Efficiency: 0.056398
Number of threads: 10, Time (in microseconds): 7789573, Efficiency: 0.059779

K-I-J ORDER
Number of threads: 1, Time (in microseconds): 32213774, Efficiency: 0.311525
Number of threads: 2, Time (in microseconds): 16208363, Efficiency: 0.156744
Number of threads: 3, Time (in microseconds): 10891695, Efficiency: 0.105329
Number of threads: 4, Time (in microseconds): 8227842, Efficiency: 0.079568
Number of threads: 5, Time (in microseconds): 6638060, Efficiency: 0.064194
Number of threads: 6, Time (in microseconds): 5556369, Efficiency: 0.053733
Number of threads: 7, Time (in microseconds): 6738563, Efficiency: 0.065166
Number of threads: 8, Time (in microseconds): 5938485, Efficiency: 0.057428
Number of threads: 9, Time (in microseconds): 5565076, Efficiency: 0.053817
Number of threads: 10, Time (in microseconds): 5428537, Efficiency: 0.052497

K-J-I ORDER
Number of threads: 1, Time (in microseconds): 78267662, Efficiency: 0.198187
Number of threads: 2, Time (in microseconds): 78204454, Efficiency: 0.198027
Number of threads: 3, Time (in microseconds): 47561783, Efficiency: 0.120435
Number of threads: 4, Time (in microseconds): 36906846, Efficiency: 0.093454
Number of threads: 5, Time (in microseconds): 31846852, Efficiency: 0.080642
Number of threads: 6, Time (in microseconds): 25924398, Efficiency: 0.065645
Number of threads: 7, Time (in microseconds): 24590123, Efficiency: 0.062266
Number of threads: 8, Time (in microseconds): 24972683, Efficiency: 0.063235
Number of threads: 9, Time (in microseconds): 24024434, Efficiency: 0.060834
Number of threads: 10, Time (in microseconds): 22618741, Efficiency: 0.057275
[pes@vandosik WorkDir]$ 
```