Frequency calculation of characters in hexadecimal alphabet

Idea of the project is to take text composed of hexadecimal alphabet characters and calculate frequency of appearance of each character. This is needed as pre-step for Huffman compression algorithm. For the solution we used openMP+MPI.
Approach for implementation of MPI is farm. Emitter is ran by process with rank 0, Collector by rank 1, while the other processes are workers.
-Emitter as input takes string, based on modifiable number of characters expected to be processed by workers (NUM_SEND constant) Emitter divides the input string and send it to available worker.
-Each worker receive segment of text and perform frequency calculation of hex characters, openMP (omp for) is used to divide the text between threads in order to provide faster calculation time. Result of each thread is stored into shared array using omp atomic. Array is then sent to Collector and Worker informs he is available for next job.
-Collector receive result array from each worker and sums the results. Final result array is then returned to main.
When all the characters of input string has been processed Emitter inform Workers by sending them special termination sign and wait for acknowledgment in order to avoid Collector terminating before all workers have finished, after all Workers have been terminated Collector also gets informed, resulting in result array being returned to main.

Detailed code explanation
We are using randomHex() function to generate String of NUMCHAR characters, NUMCHAR is defined as constant.

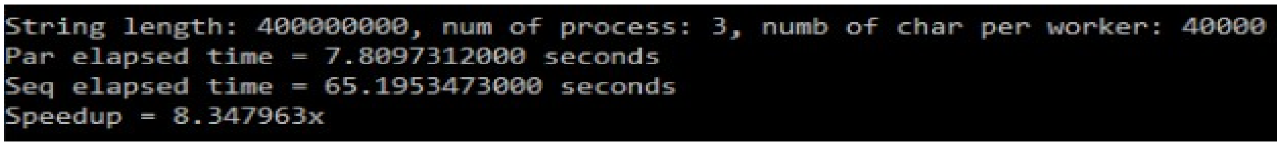
How does Emitter work?
Generated string is taken as input. Depending on NUM_SEND, the number of characters expected to be processed by each worker, text is divided into strings of NUM_SEND characters while the last string, considering NUMCHAR%NUM_SEND !=0 is filled with ‘0’ to match NUM_SEND. (int)brojac is used to keep track of which segment is next to be sent to available worker. When there are no more segments to send workers are informed by termination sign (string starting with ‘0’), after the acknowledgment that all workers are finished Emitter also inform Collector.

How does Worker work?
Worker receive input string from Emitter that needs to be processed. Using “omp for” work is divided to number of threads specified in num_threads(). Result of each thread is stored into shared array using omp atomic. In case the received string starts with ‘0’ worker returns and informs Emitter he is terminated.

How does Collector work?
Collector receive array[15] from any source. If its from worker he sums it up with the result array (resAll) but in case its from Emitter, that means the job is finished and resAll is returned to main.

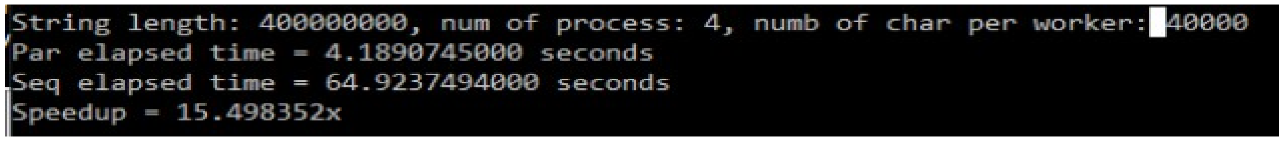
Measurements and perfomance

1. String length 400000000 number of char per worker 40000
Number of workers depends on number of processes, with only 3 processes there is only one worker and measurement in that case is showed in picture below:

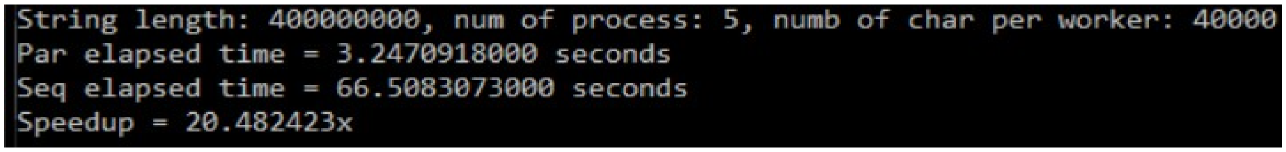


```
String length: 400000000, num of process: 3, numb of char per worker: 40000
Par elapsed time = 7.8097312000 seconds
Seq elapsed time = 65.1953473000 seconds
Speedup = 8.347963x
```

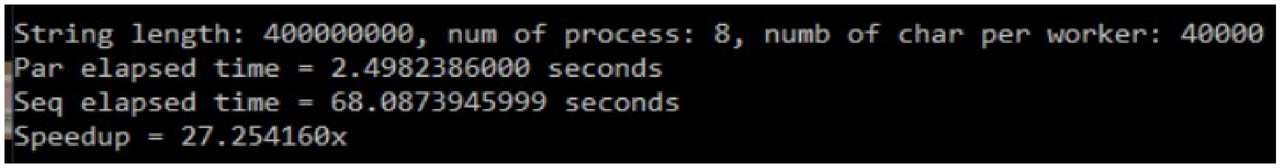
Parallel version even though it has only one worker perform 8.35x faster than Seq version, that is because worker is implemented to do parallel processing of the input string he receive.
With same String length and numb of char per worker but with 4,5,8 processes measurements are:



```
String length: 400000000, num of process: 4, numb of char per worker: 40000
Par elapsed time = 4.1890745000 seconds
Seq elapsed time = 64.9237494000 seconds
Speedup = 15.498352x
```



```
String length: 400000000, num of process: 5, numb of char per worker: 40000
Par elapsed time = 3.2470918000 seconds
Seq elapsed time = 66.5083073000 seconds
Speedup = 20.482423x
```

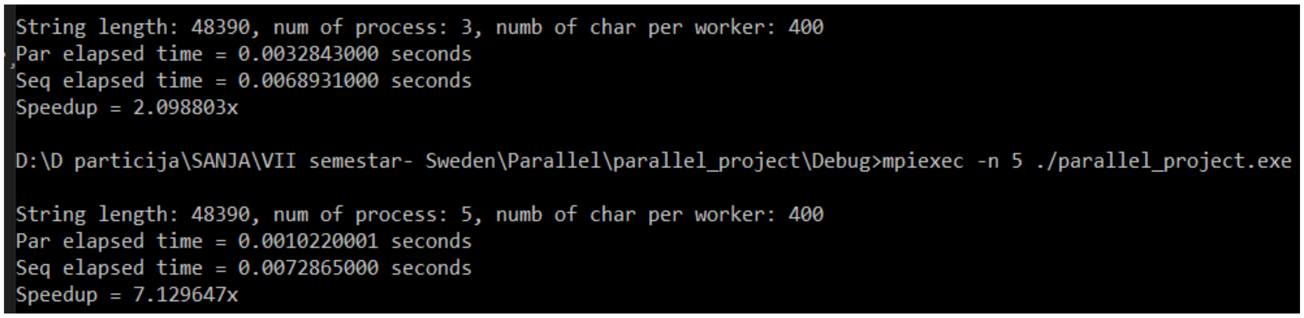


```
String length: 400000000, num of process: 8, numb of char per worker: 40000
Par elapsed time = 2.4982386000 seconds
Seq elapsed time = 68.0873945999 seconds
Speedup = 27.254160x
```

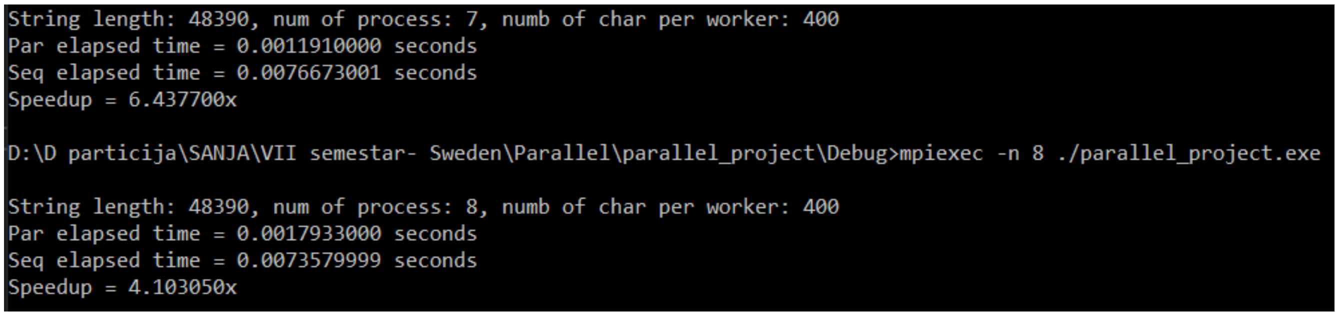
Higher amount of processes means more workers will work in same time resulting in higher execution time.

Note: Time of sequential is not constant, it depends on the String generated, string generated is different every time and its processed by Switch statement resulting in minor time differences.

2. String length 48930 number of char per worker 40000000



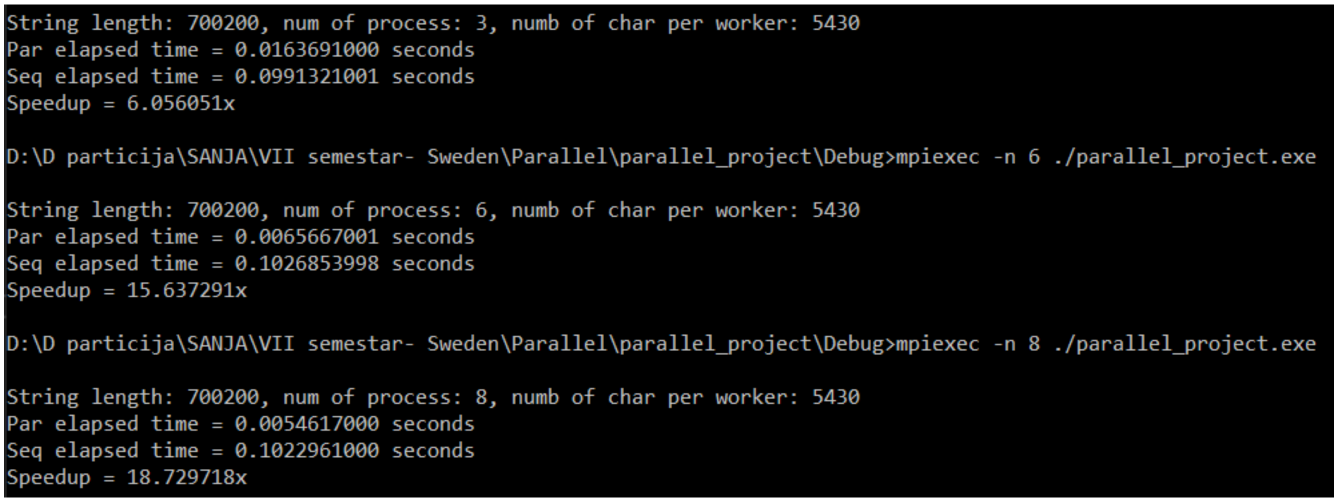
```
String length: 48390, num of process: 3, numb of char per worker: 400
Par elapsed time = 0.0032843000 seconds
Seq elapsed time = 0.0068931000 seconds
Speedup = 2.098803x

D:\D particija\SANJA\VII semestar- Sweden\Parallel\parallel_project\Debug>mpiexec -n 5 ./parallel_project.exe

String length: 48390, num of process: 5, numb of char per worker: 400
Par elapsed time = 0.0010220001 seconds
Seq elapsed time = 0.0072865000 seconds
Speedup = 7.129647x
```



```
String length: 48390, num of process: 7, numb of char per worker: 400
Par elapsed time = 0.0011910000 seconds
Seq elapsed time = 0.0076673001 seconds
Speedup = 6.437700x

D:\D particija\SANJA\VII semestar- Sweden\Parallel\parallel_project\Debug>mpiexec -n 8 ./parallel_project.exe

String length: 48390, num of process: 8, numb of char per worker: 400
Par elapsed time = 0.0017933000 seconds
Seq elapsed time = 0.0073579999 seconds
Speedup = 4.103050x
```

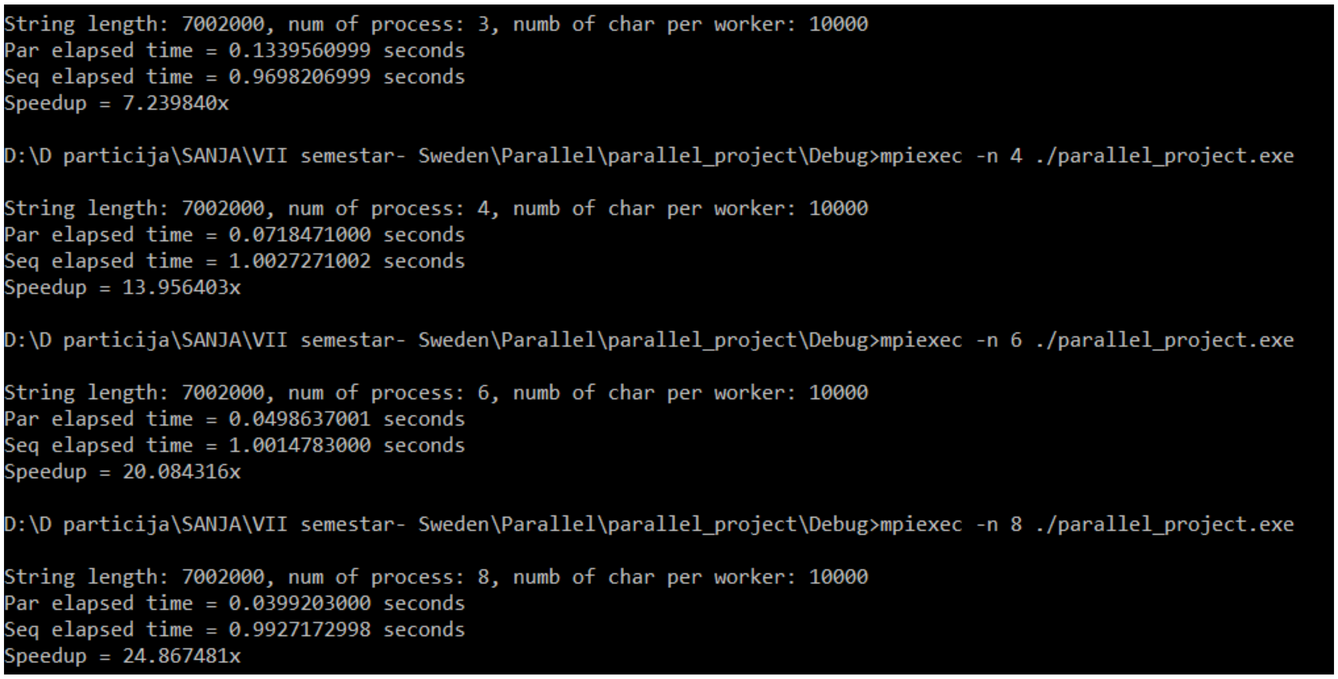
As shown above, each workers does simple job meaning with higher amount of workers Speedup will eventually decrease because of the time required to manage the workers.

3. String length 700200 number of char per worker 5430



```
String length: 700200, num of process: 3, numb of char per worker: 5430
Par elapsed time = 0.0163691000 seconds
Seq elapsed time = 0.0991321001 seconds
Speedup = 6.056051x

D:\D particija\SANJA\VII semestar- Sweden\Parallel\parallel_project\Debug>mpiexec -n 6 ./parallel_project.exe

String length: 700200, num of process: 6, numb of char per worker: 5430
Par elapsed time = 0.0065667001 seconds
Seq elapsed time = 0.1026853998 seconds
Speedup = 15.637291x

D:\D particija\SANJA\VII semestar- Sweden\Parallel\parallel_project\Debug>mpiexec -n 8 ./parallel_project.exe

String length: 700200, num of process: 8, numb of char per worker: 5430
Par elapsed time = 0.0054617000 seconds
Seq elapsed time = 0.1022961000 seconds
Speedup = 18.729718x
```

4. String length 7002000 number of char per worker 10000



```
String length: 7002000, num of process: 3, numb of char per worker: 10000
Par elapsed time = 0.1339560999 seconds
Seq elapsed time = 0.9698206999 seconds
Speedup = 7.239840x

D:\D particija\SANJA\VII semestar- Sweden\Parallel\parallel_project\Debug>mpiexec -n 4 ./parallel_project.exe

String length: 7002000, num of process: 4, numb of char per worker: 10000
Par elapsed time = 0.0718471000 seconds
Seq elapsed time = 1.0027271002 seconds
Speedup = 13.956403x

D:\D particija\SANJA\VII semestar- Sweden\Parallel\parallel_project\Debug>mpiexec -n 6 ./parallel_project.exe

String length: 7002000, num of process: 6, numb of char per worker: 10000
Par elapsed time = 0.0498637001 seconds
Seq elapsed time = 1.0014783000 seconds
Speedup = 20.084316x

D:\D particija\SANJA\VII semestar- Sweden\Parallel\parallel_project\Debug>mpiexec -n 8 ./parallel_project.exe

String length: 7002000, num of process: 8, numb of char per worker: 10000
Par elapsed time = 0.0399203000 seconds
Seq elapsed time = 0.9927172998 seconds
Speedup = 24.867481x
```

Aleksandra Ristic, Nikola Jovic