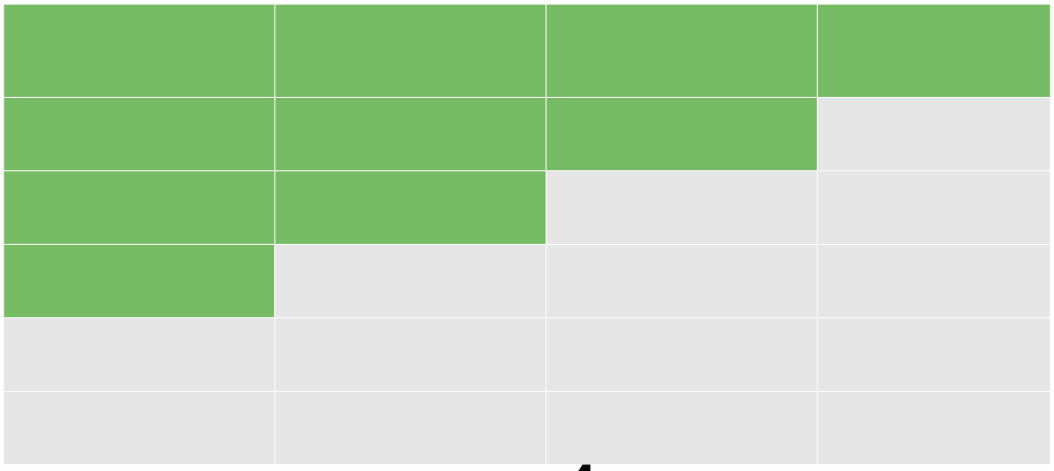
gevochobanyan@gmail.com
gc401929

Bottomless Bag Test

In the sequential approach, we just use standard dynamic programming. We keep a 2-dimensional array with a size of **W**x**N** where **W** is the weight of the bag and **N** is the number of elements that we have. We traverse the array row by row, left to right and refer to previous answers in order to get the current one.

In the parallel approach, we split the whole array into **n** columns where **n** is the number of threads.
Here is how we minimize the waiting for the threads which are responsible for the columns closer to the right side. We need to make sure that on a given row thread **j** has finished its portion before the thread **j+1** can start its portion on the same row. Keeping this in mind we take the undermentioned approach.

Thread **0** computes its portions till row **n**-1 and every time it finishes a portion of a row, thread **1** is allowed to start its portion on that row. Similarly thread **j** computes its portions till row **n-1-j** and every time it finishes a portion of a row, thread **j+1** is allowed to start its portion on that row. When this process finished result looks something like this:

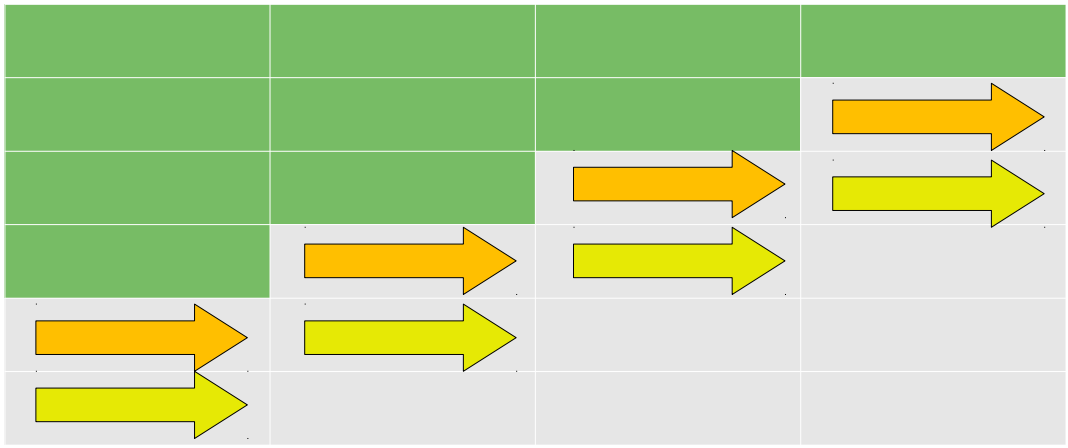


n=4

Now each all threads can start working on the cells on the diagonal at the same time. As soon as one thread finishes its cell it has to wait for all the threads to finish their cells on the diagonal so that each of them can go to next row.

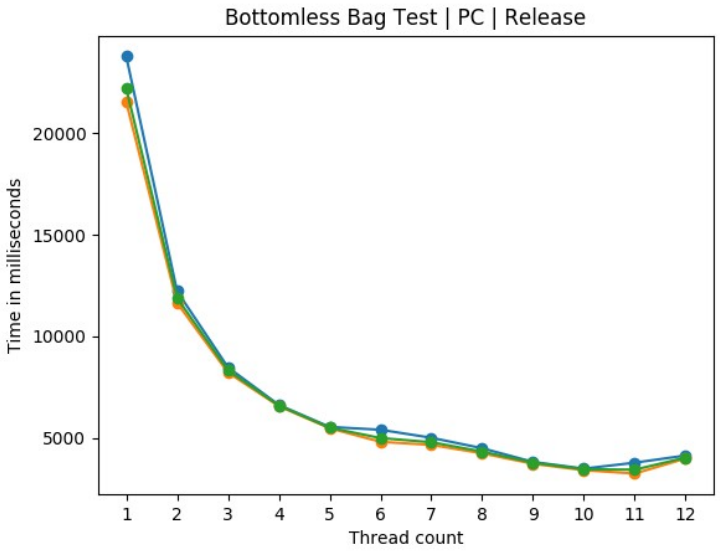
The following picture shows the aforementioned approach. First the orange arrows execute, then after the last arrange arrow finishes, all the yellow arrows start their execution. It goes on like this until all of the cells are computed.

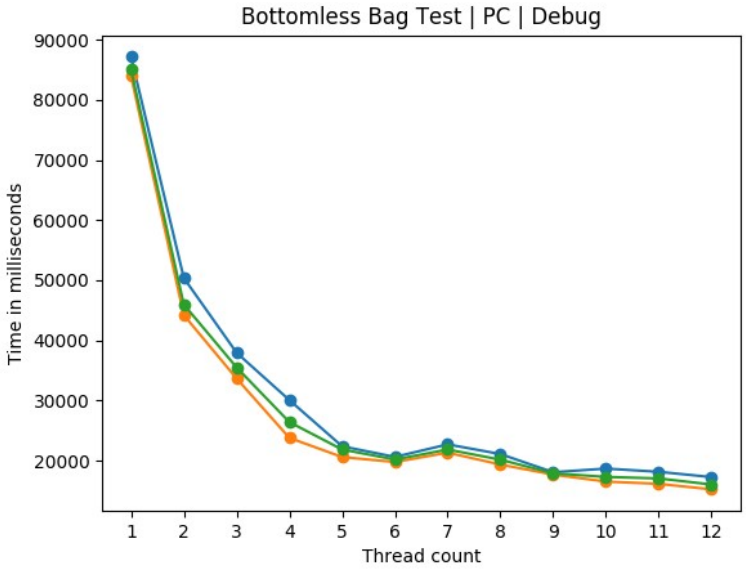
gevochobanyan@gmail.com
gc401929

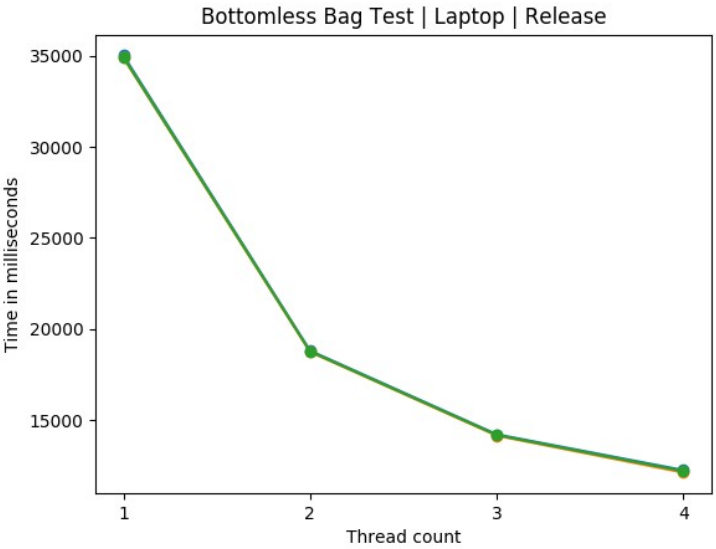


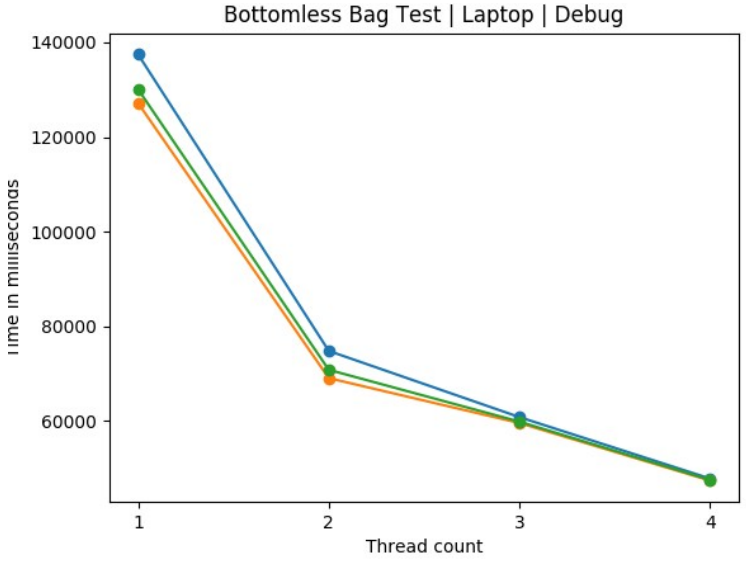
Below are benchmark tests ran on 2-core 4-thread Laptop and 6-core 12-thread PC.
The graphs show the performance gains when increasing the thread count. The reason why we see a performance boost only with the next even thread, is that we split the array into two halves.

•Each test is repeated 5 times and minimum, maximum, and average is taken.









Concurrent Programming Project
Gevorg Chobanyan
gevochobanyan@gmail.com
gc401929

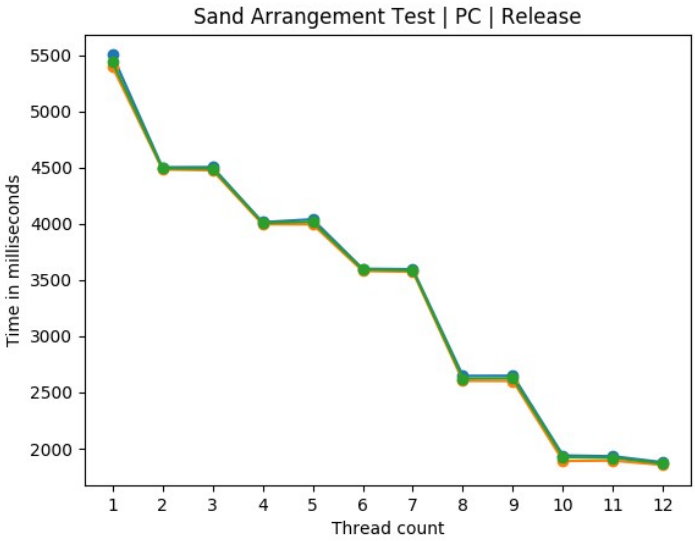
Sand Arrangement Test

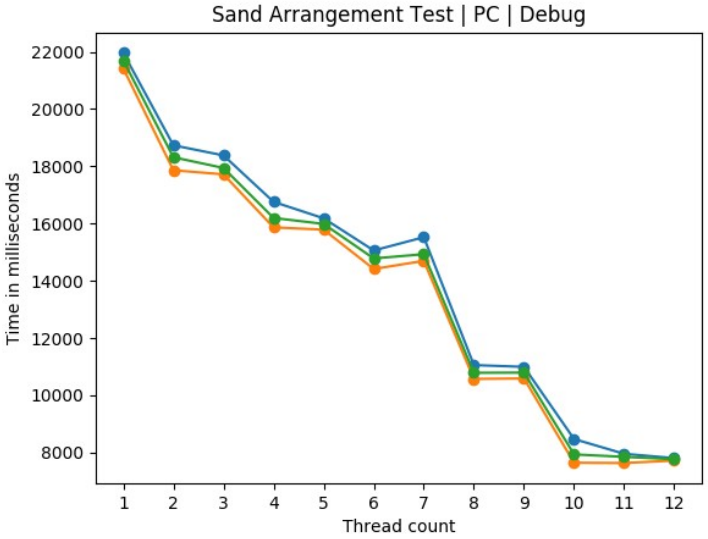
The utilized sorting algorithm is QuickSort. First our current thread finds the pivot for the given array. Then, all there is left to do is to sort the halves indicated by the pivot. Our current thread does the second half and leaves the second half to some other thread if one is available.

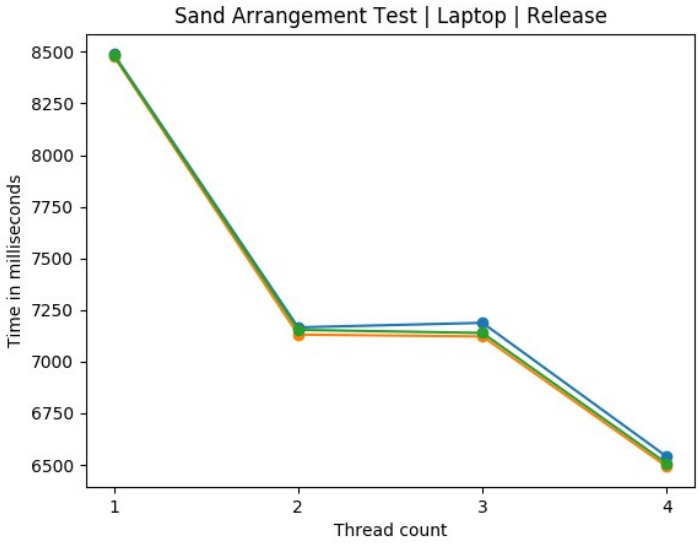
The Reason I chose QuickSort instead of MergeSort is that with MergeSort after concurrently sorting **n** halves of the array it is still needed to merge those sorted part which can only be done sequentially, leaving the rest of the threads unutilized.

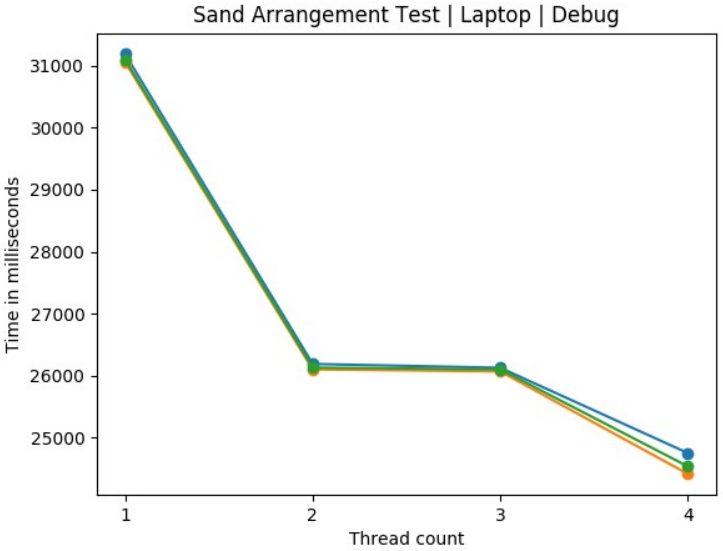
Below are benchmark tests ran on 2-core 4-thread Laptop and 6-core 12-thread PC.
The graphs show the performance gains when increasing the thread count. The reason why we see a performance boost only with the next even thread, is that we split the array into two halves.

- Each benchmark is ran on an array of random integers, with a size of 1000000
- Each test is repeated 5 times and minimum, maximum, and average is taken.









Concurrent Programming Project
Gevorg Chobanyan

gevochobanyan@gmail.com
gc401929

Crystal Selection Test

Finding the maximum value in the array is achieved through splitting the array into **n** equal parts where **n** is the number of threads that we have. Once that is done each thread simply finds the maximum value from its segment and gives it back. Once every thread finds its maximum value, we pick the maximum among them.

Below are benchmark tests ran on 2-core 4-thread Laptop and 6-core 12-thread PC.
The graphs show the performance gains when increasing the thread count. We can see that this approach scales relatively well.

- Each benchmark is ran on an array of random integers, with a size of 100000000
- Each test is repeated 5 times and minimum, maximum, and average is taken.