Lucien Iseli, Loris Pilotto
274999, 262651

Assignement 2

1)To optimize *simulate* we parallelized the loop iterating on the lines of the two dimensional array paying attention to not create new threads for every of the N iterations of the algorithm because it's a big waste of time since we would create new threads N times . We also gave each thread spatially close lines to compute (i.e. using the static for parallelization which give to the ith thread the lines i(#lines/#threads) to (i+1)((#lines/#threads)-1 to compute). We splitted the lines this way because we want to optimise the hit ratio. Finally, we added a barrier before swapping the variables « input » and « output » to be sure to have valid output (we don't want to swap the input and output before all the threads have finished computing their lines).

The tree main way to split the work among threads are:

-Give the next line to compute to the first available thread, but it's not very efficient since we don't exploit locality because the computation of the output depends on multiple rows. The number of misses will be high and the core processor will have to wait the data to come from the memory.

-Give to each thread a block of spacialy close lines (what we did in our work). It's better since we exploit the locality so there will be less miss so it's a better optimisation.

-The last way is to give each thread a square of the two dimensional array (blocking). The goal is also to give each thread spacialy close data. It's theoretically more efficient than a sequence of lines because the whole block would be in the cache and we reuse each element of the input multiple time. But with our algorithm the execution time was strangly longer with this method than by separating the two dimensional array in spacialy close lines, this is probably due to more loop overhead, because the code gets significantly more complex.

So what we did to optimize the memory:

-We read the data by line and not by row to optimize hits

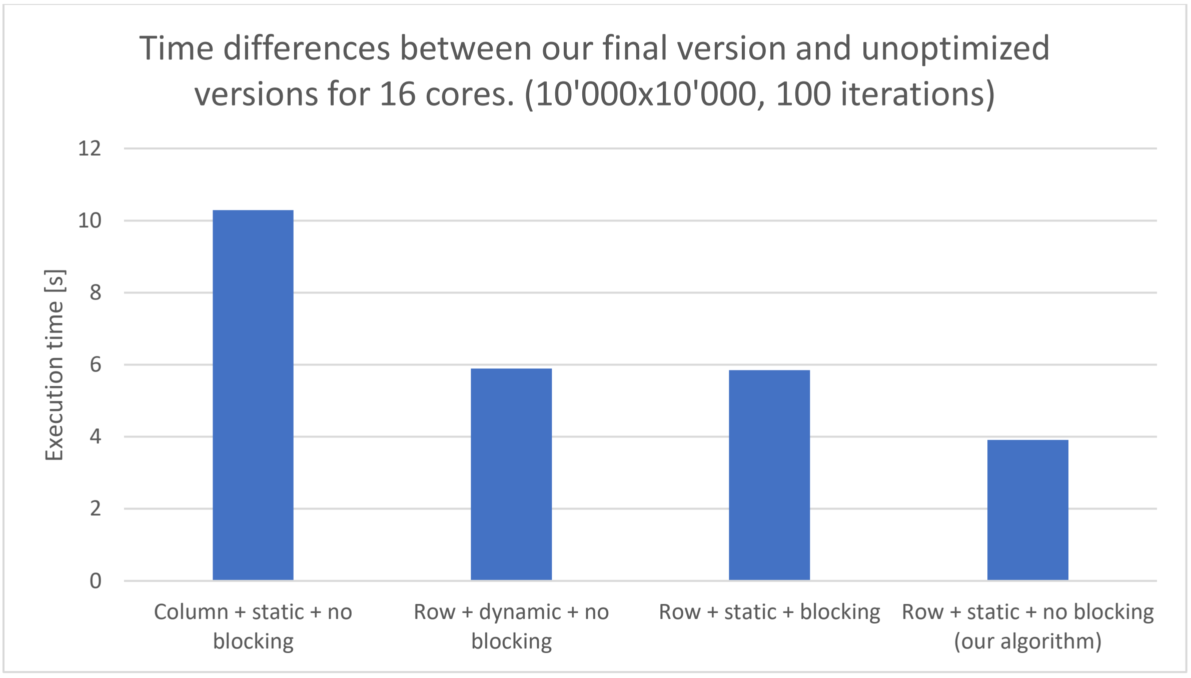
-Use the schedule static for the loop to optimize hits with spatiality

-We have only one fork and join because the data for every core is well split

Finally, we can notice that there is a small issue with load balancing because the last thread can have a bigger block than the other threads due to static scheduling and therefore a waste of time before the barrier, but the difference in the block size is not significant (it depends on the number of threads and number of lines).

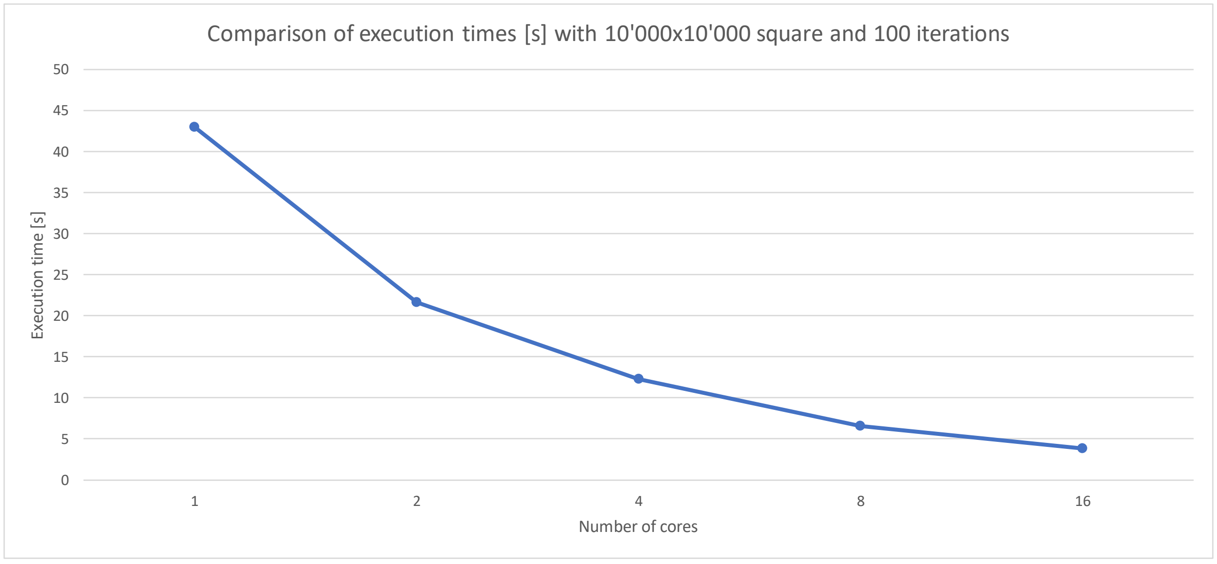
Lucien Iseli, Loris Pilotto
274999, 262651

2)



Here are the different time executions for our program (to the right) if we remove one of the major optimizations we added, and we also added the time if we added blocking. We are not sure why blocking makes it worse but this is probably due to more loop overhead. All the code is running with a 10’000 x 10’000 square with 100 iterations on 16 cores. We isolated each optimization by removing only one of them for each execution time, so the result should be reasonably close to reality. We can see that cache hits are a really big deal because doing the algorithm by columns or by rows completely changes the execution time. But also dynamic vs static this is due to the spatiality of the lines each core is assigned to.

3)



Here are the execution times on the cluster for {1, 2, 4, 8, 16} cores. We can see that we obtained a very reasonable speedup! With 16 cores the code is 11.23x faster than with 1 core.