“Life in Cilk”

Antonio Javier Samaniego Jurado
Pablo Martín García

Introduction

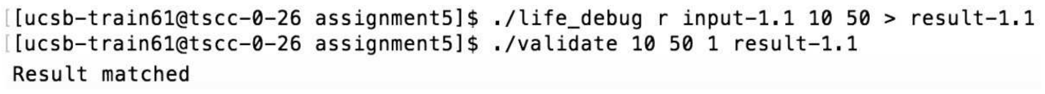
The purpose of this assignment is to implement a cellular automaton called the Game of Life in Cilk, and to tune it to get maximum performance.

Procedure

After using Cilk (in particular *cilk_for*) to implement the Game of Life, we have reported the speedup, efficiency, and Cells Updated Per Second (CUPS).

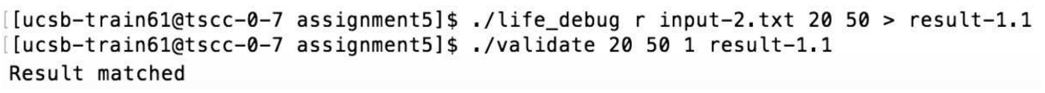
We want to point out that the code provided (submit.cpp) does work (that is, we have successfully checked its correctness) when, for some reason, we replace the *cilk_for* loop with a normal/standard *for* loop, as we can see in the following output examples (using different input files):

Output 1:



```
[ucsb-train61@tscc-0-26 assignment5]$ ./life_debug r input-1.1 10 50 > result-1.1
[ucsb-train61@tscc-0-26 assignment5]$ ./validate 10 50 1 result-1.1
Result matched
```

Output 2:



```
[ucsb-train61@tscc-0-7 assignment5]$ ./life_debug r input-2.txt 20 50 > result-1.1
[ucsb-train61@tscc-0-7 assignment5]$ ./validate 20 50 1 result-1.1
Result matched
```

This was very weird for us and, even after struggling a lot debugging the code looking for bugs and discussing the issue with some classmates, we could not find out why this was happening.

However (and this is why we have been able to do the report), we are pretty sure (and agree with our classmates) that the issue has to do with overlapping memory values, which is why when in the code we use the *cilk_for* the numerical results do not match, but the running times seem to be reasonable.

For the speedup, we have used:

num_cores = [1 2 3 4 8 12];

We have shown the results in Fig. 1.

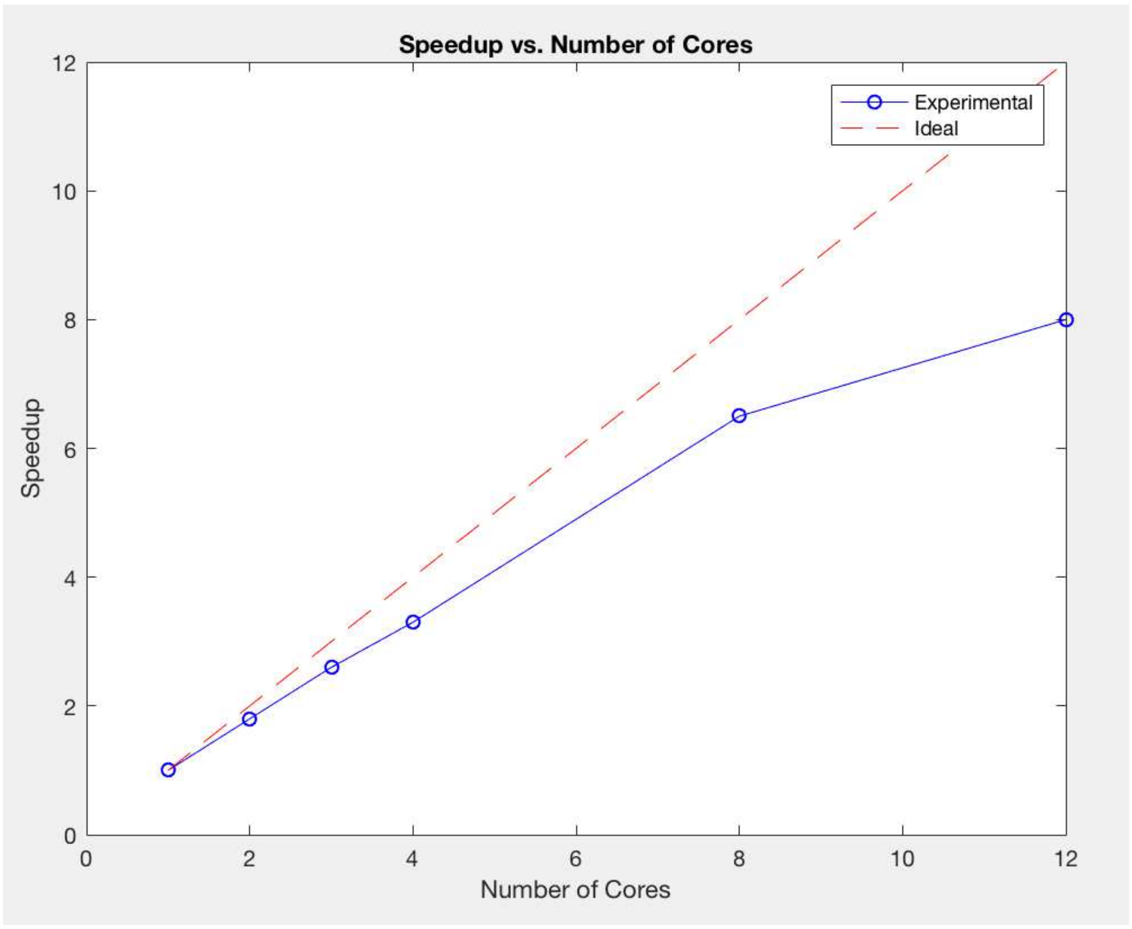


Fig. 1. Speedup vs. Number of Cores

We can see how, although it is not linear, as commonly expected in experimental performances the trend is not exactly linear (sub-linear), which is a reasonable result in practice for a decent performance.

In order to measure the efficiency, we have divided the speedup values by the correspondent number of cores values. The results can be seen in Fig 2.

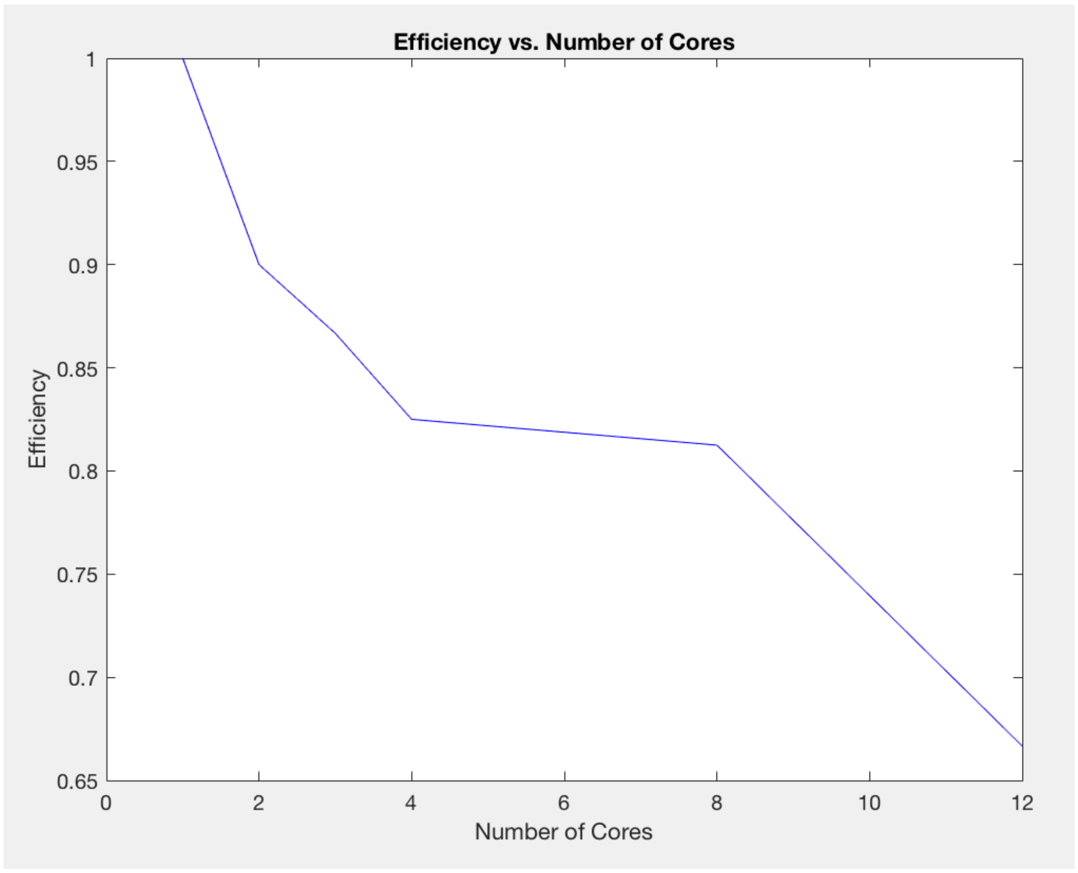


Fig. 2. Efficiency vs. Number of Cores

As we can see, the parallel efficiency goes down towards the point diminishing returns as we increase the number of cores, as opposed to what the theoretical/ideal results would show. As we said in other assignments, again, this tells us how many processors we should use in order to get an optimal result.

An ideal/theoretical efficiency would be equal to 1 for all number of cores, but this result is reasonable considering we are carrying out experimental measurements in a real case.

To measure the number of Cells Updated Per Second (CUPS) against *n*, we have used the following values to set the matrix size:

Matrix Size (n) = [400 500 1200 2500]

We have shown the results in Fig. 3.

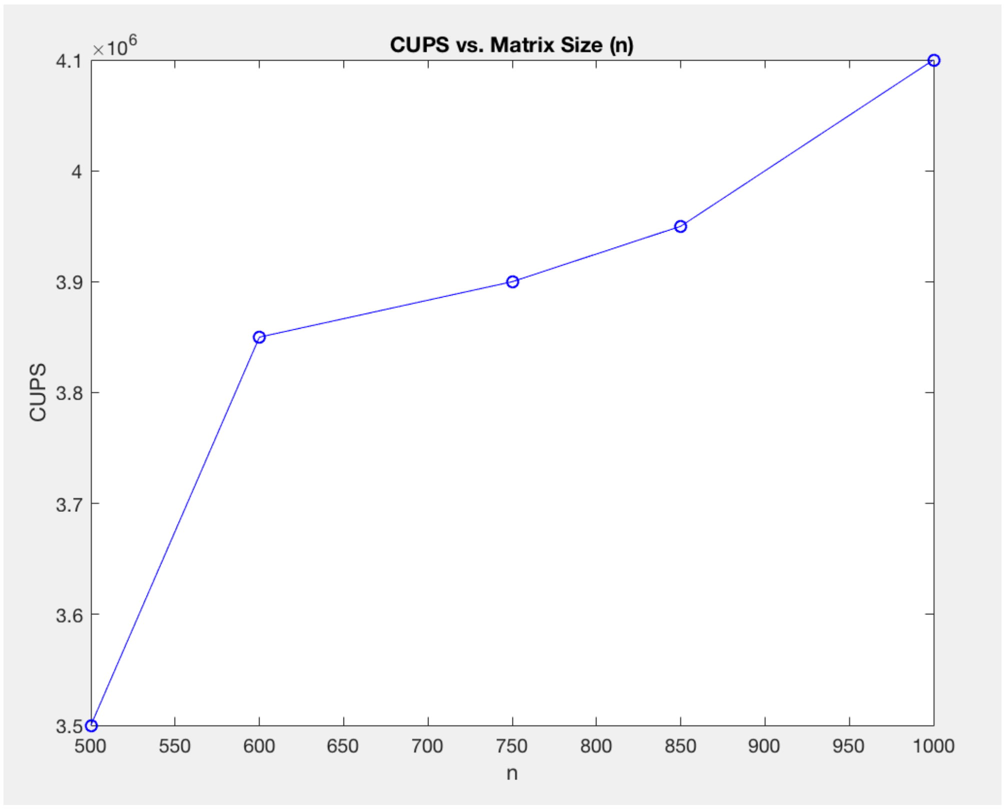


Fig. 3. CUPS vs. Matrix Size (n)

We can see how the trend is an increasingly growing function, since it initially takes more time to be computed.

It should be pointed out that we have used values for the matrix size *n* that could give us results in a reasonable amount of time (about 1-4 minutes aprox. in our case), for as much as 1000 iterations, for large boards (that is, large matrix sizes) that go from n=500 to n=1000. From n=1000 and on, the trend was pretty much similarly constant.

<u>Conclusion</u>

The potential of the Game of Life can successfully be implemented in C++ using Cilk, so that parallelizing the algorithm we can get reasonably decent values of running time, and consequently CUPS (as much as about 4 CUPS), for computations that use/require pretty large boards, which again shows the potential of Cilk in order to compute data and tasks more efficiently.