Game of Life - HW #5

Sverre Kvamme &Simen Andresen

February 24, 2013

Explanation of Code

A little note on notation: whenever "rows of a" are used, it indicates the rows of the board stored in the array a.
submit.cpp::life() - This function starts out be making a temporary buffer called a˙temp, that is used for counting neighbours. in each iteration this buffer's pointer is swapped with a which is the buffer written to, so that a_temp is always the updated version of the cells without having to copy over to buffer each iteration. Also before the iterations start, all rows of a are checked for any 1-entries, and this information is stored for row i into $row\_has\_ones[i]$. A parallized for loop - cilk˙for - is used to iterate through all rows of a and if none of the previous, current or next row has any 1- entries it continues to the next row. When a 1-entry is found it iterates through all columns of the row, counts the neighbour of all cells, and decides wether to kill or to make new life. It also updates the $row\_has\_ones[i]$ accordingly.

Running and Validating the code

In the harness, only the Makefile has been changed where the only change is adding a -O3 option to optimize the C++ code. The compilation, running and validation is otherwise the same as in the Readme.txt

Performance

For performance testing the -O3 option was omitted. From Fig. 1 it can be seen that the number of CUPS increases for increasing n. This can be due to the fact that each row not containing any 1's will be skipped, since our program does not actually process rows that does not have neighbour rows with 1-entries. The parallel efficency is relatively good for 2 to 8 processors, but is slightly decreasing for an increase in number of processors. Also the parallel efficiency is varying for different size of n, which can be due to the fact that the data size is increasing quadraticly with n, and this can have effect on the cache.

Conclusion

With our strategy we acheived a suprisingly fast runtime, and the parallel efficiency was very satisfactory. It must be noted that our algorithm works best for scarse game boards (many 0s compared to 1s), since it skips all rows not neighbouring rows with 1-entries.

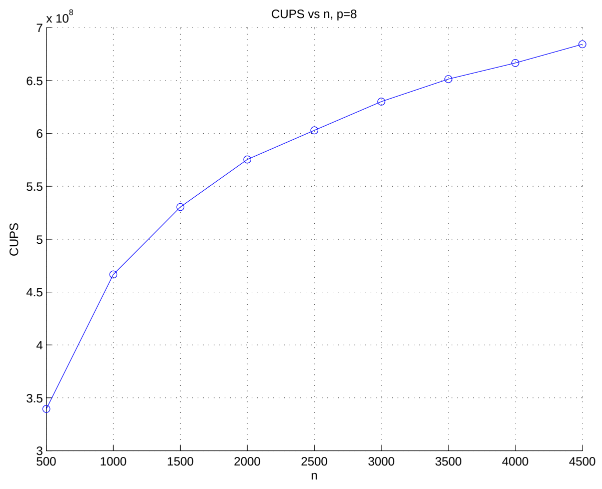

Figure 1: CUPS vs. n for 1000 iterations, running on 8 cores

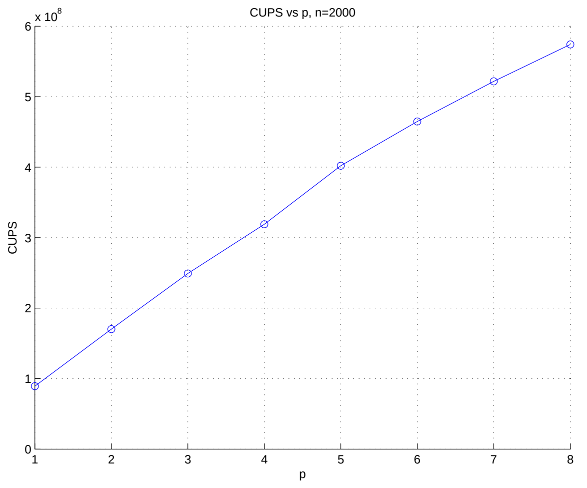

Figure 2: CUPS vs. processor for 1000 iterations, and n=2000

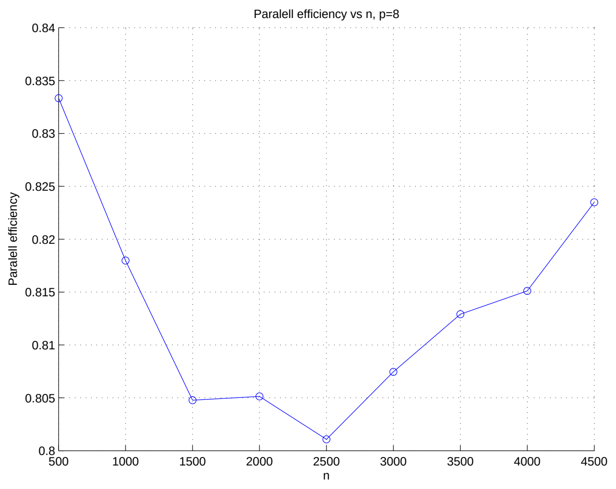

Figure 3: Parallel efficiency vs. n for 1000 iterations running on 8 cores

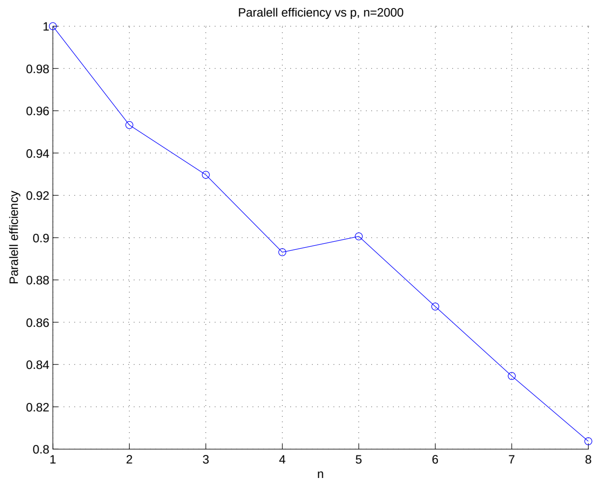

Figure 4: Parallel efficiency vs. processor for 1000 iterations and n=2000