HPC - 4. Assignment - Solving 2D Gray-Scott Model with MPI

Kristjan Kostanjšek, Nejc Ločičnik

I. Introduction

The Gray-Scott model is a well-known example of a reaction-diffusion system, which describes how the concentration of chemical substances changes over space and time due to local reactions and diffusion. In this report, we implement a simulation of the Gray-Scott model to observe the formation of complex patterns over a two-dimensional grid, example of such pattern can be observed on Figure 1. We first developed a sequential version of the program to establish a correct and reliable baseline. Afterwards, we parallelized the program using MPI, exploring different work distributions, measured and analyzed the performance improvements achieved by these optimizations, comparing the speedups of the parallelized implementations against the original sequential version. This report presents the implementation details, results, and analysis of our findings.

Figure 1. Example of a pattern generated by our Gray-Scott model.

II. Optimizations and Results

In this section we will present the results of the baseline program and the optimized versions, compare their execution times and analyze them.

A. Sequential Version

First we created the sequential version of the program. We ran the program on 5 different grid sizes, each run was limited to 5000 simulation steps. The execution times (in seconds) depending on the grid size are presented in the Table I. The measured execution time includes only the Gray-Scott simulation itself, with grid initialization excluded from the timing.

grid size	256x256	512x512	1024x1024	2048x2048	4096x4096
$t_S[s]$	13.08	46.56	182.36	721.63	2882.33

Table I
Execution times for sequential program.

B. Row-wise work distribution with MPI

The first attempt of parallelizing the Gray-Scott simulation using MPI was by distributing the work by rows, i.e. splitting the grid into horizontal strips, assigning each strip to a process and utilizing MPI to exchange the so-called *ghost rows* between the processes. The program was once again run on 5 different grid sizes for 5000 simulation steps. We experimented with different number of cores, ranging from 1 to 64. The speedups (measured as $\frac{t_S}{t_P}$) can be observed in Table II.

grid size	256x256	512x512	1024x1024	2048x2048	4096x4096
1 core	1.90	1.88	1.94	1.96	1.97
2 cores	2.20	2.02	2.02	2.01	2.01
4 cores	3.65	3.88	3.95	4.00	4.01
16 cores	8.96	12.97	14.97	15.46	15.74
32 cores	15.21	21.86	27.93	30.24	30.35
64 cores	26.16	34.75	50.77	57.92	47.97

Table II
Speedups for row-wise work distribution with MPI for different numbers of cores.

In the results shown in Table II, we can observe that increasing the number of cores generally leads to a corresponding increase in speedup, approaching approximately $n\times$ as the number of cores n increases. This trend is most evident on the larger grids, where the computational workload is sufficient to offset the overhead of inter-process communication.

In theory, the speedup should not exceed the number of cores due to the added cost of communication between processes. However, in our results, we occasionally observe speedups slightly exceeding the expected limit for smaller core counts. This could be attributed to measurement variability or insufficient averaging across multiple runs, which would help smooth out transient fluctuations in system load or network performance.

An unexpected and interesting anomaly occurs when using a single core: the measured speedup is still comparable to that achieved with two cores. In principle, this should not happen, as the sequential baseline and the one-core parallel version should perform similarly. One possible explanation for this behavior could be differences in how the MPI implementation handles process setup and memory management, even when confined to a single process. It is also possible that the overhead of initializing the MPI environment introduces subtle variations in timing measurements. Further investigation with more extensive benchmarking and profiling would be needed to conclusively identify the cause.

III. Conclusion

TODO.