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

Kristjan Kostanjšek, Nejc Ločičnik

I. Introduction

The Gray-Scott model serves as a canonical example of reaction-diffusion systems, simulating the emergence of complex spatial patterns through the interplay of chemical reactions and diffusion processes. This computationally intensive model exhibits excellent potential for parallelization, as each grid point's evolution depends solely on its local neighborhood, enabling efficient domain decomposition. Figure 1 illustrates the characteristic Turing patterns generated by our implementation, demonstrating the model's capacity to produce intricate structures from simple local interaction rules.

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

Our systematic investigation evaluates multiple parallelization strategies through MPI, achieving significant performance improvements across various problem scales. The most effective configuration demonstrates up to 218× speedup for 2048×2048 grids using block-wise decomposition, with particularly strong scaling for medium-sized problems. These results underscore how careful consideration of data layout and communication patterns can substantially enhance parallel efficiency in scientific simulations.

II. Optimizations and Results

The following subsections describe our experimental approach to optimizing the sequential Gray-Scott simulation code and analyze the achieved speedups. Beginning with the baseline sequential implementation, we conducted four key experiments: (1) MPI parallelization with row-wise distribution, (2) MPI with reduced communication frequency through increased halo boundaries, (3) comparison of single-node versus multi-node MPI performance, and (4) block-wise distribution utilizing MPI derived datatypes.

The speed-up is calculated using the formula: $S = t_S/t_P$, where t_S represents the execution time of the sequential program, and t_P denotes the execution time of the parallel program.

A. Sequential Version

We first established a baseline by implementing and benchmarking the sequential version across five grid sizes, running each configuration for 5000 simulation steps. Table I presents the execution times (in seconds), which measure only the core Gray-Scott computation while excluding grid initialization overhead.

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

Our initial parallelization strategy employed MPI with row-wise domain decomposition, dividing the computational grid into horizontal strips assigned to individual processes. This approach required periodic exchange of boundary rows between neighboring processes. We evaluated performance across the same five grid sizes using 1 to 64 cores, with results shown 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
Row-wise work distribution speedups (1 node).

The row-wise MPI implementation shows sublinear scaling across all configurations, with performance improving for larger grids where computation outweighs communication costs. Notably, the single-core parallel version performs similarly to the 2-core case (1.90-2.20 speedup), likely due to MPI overhead. While speedups increase with core count (reaching 57.92 for 2048x2048 at 64 cores), the approach remains communication-bound, motivating our exploration of alternative decomposition strategies. The results confirm the need for more efficient data distribution methods to overcome these limitations.

C. Optimizing Communication Frequency Through Extended Halo Regions

This experiment roughly examined the trade-offs between communication frequency and halo region size in our MPI implementation. By expanding the boundary halo width from 1 to 5 rows and reducing communication frequency to every fifth iteration, we aimed to decrease synchronization overhead while maintaining computational accuracy. The approach hypothesized that the reduced number of larger data exchanges would compensate for the additional computational work required to process wider halo regions.

grid size	256x256	512x512	1024x1024	2048x2048	4096x4096
2 core	4.22	3.91	3.94	3.95	2.01
4 cores	3.60	3.83	3.92	3.94	3.98
16 cores	8.23	11.79	14.19	15.11	15.52
32 cores	10.72	19.48	25.29	28.44	29.39
65 cores	16.77	28.22	41.63	51.25	49.08

Table III
Speedups of row-wise work distribution with reduced communication frequency (1 node).

The results shown in Table III, demonstrate that while the extended halo approach reduces communication frequency, it introduces memory bandwidth limitations in shared-memory MPI configurations. For smaller core counts (2-4 cores), performance remains comparable to the standard approach as

memory bandwidth suffices. However, at higher core counts (16+ cores), wider halo regions cause noticeable performance degradation due to increased memory contention during bulk transfers. This suggests that in shared-memory environments, frequent small messages often outperform infrequent large exchanges, as they better match the memory subsystem's characteristics. The trade-off becomes particularly evident with larger grid sizes, where memory bandwidth saturation limits potential gains from reduced communication frequency.

D. Distributed Memory Performance Analysis: Single-Node vs. Multi-Node Execution

This investigation evaluates the performance characteristics of our MPI implementation when distributed across single-node and multi-node configurations, revealing fundamental differences in scaling behavior based on memory architecture. The comparative analysis focuses on how shared-memory communication within a single node contrasts with network-based communication across nodes, particularly examining how these differences affect performance for varying problem sizes and core counts.

grid size	256x256	512x512	1024x1024	2048x2048	4096x4096
2 core	2.58	3.27	3.72	3.85	3.92
4 cores	3.35	3.62	3.92	3.98	4.01
16 cores	7.31	12.48	14.48	15.44	15.83
32 cores	8.78	19.40	27.26	29.35	31.03
64 cores	10.46	26.91	44.70	57.87	59.40

Table IV
Row-wise work distribution speedups (2 nodes).

The multi-node results shown in Table IV reveal two distinct scaling behaviors. For large problems (2048x2048+) at high core counts (32-64 cores), distributed execution achieves superior performance (up to 59.40 speedup) by alleviating memory bandwidth constraints through dedicated network bandwidth. However, smaller problems (256x256) with few cores show limited benefits (2.58 speedup) as network latency outweighs computational gains. This dichotomy suggests adaptive strategies could optimize performance by dynamically selecting between shared-memory and distributed approaches based on problem size and core count. The results highlight how memory-bound and communication-bound regimes require different optimization strategies in parallel implementations.

E. Block-Wise Domain Decomposition with MPI Derived Data Types

This investigation evaluates an optimized parallelization strategy employing block-wise domain decomposition enhanced by MPI derived data types. The approach fundamentally restructures the data layout from row-wise strips to contiguous rectangular blocks, aiming to simultaneously improve cache utilization and reduce communication overhead. By leveraging MPI's derived datatype functionality, we optimize boundary exchange patterns while maintaining the algorithm's numerical correctness.

grid size	256x256	512x512	1024x1024	2048x2048	4096x4096
2 core	9.28	9.76	6.65	5.41	4.89
4 cores	9.98	16.81	20.40	10.41	7.55
16 cores	23.36	36.95	57.35	44.33	21.50
32 cores	29.07	55.43	96.49	111.71	49.88
64 cores	48.44	87.85	162.82	218.02	52.11

Table V
Block-wise work distribution speedups with derived data types (1 node).

The block-wise decomposition shows (Table V) substantial performance gains, delivering up to 3.45× improvement (96.49 vs 27.93 speedup) for medium grids by optimizing cache locality and communication patterns. Even with just 2 cores, better data organization alone yields 2-4× speedups. However, memory bandwidth limits emerge for the largest grid (4096x4096) at 64 cores (52.11 speedup), suggesting future work should combine block-wise with multi-node distribution. While cluster constraints prevented testing this hybrid approach, the results clearly demonstrate the value of optimized data layouts in MPI implementations.

III. Conclusion

Our comprehensive optimization of the Gray-Scott reaction-diffusion model has demonstrated the effectiveness of MPI parallelization for scientific simulation. The study reveals three key findings: (1) Block-wise decomposition with MPI derived datatypes delivers superior performance (up to 3.45× improvement over row-wise approaches), (2) Memory bandwidth emerges as the critical bottleneck for single-node implementations at scale, and (3) Multi-node execution becomes essential for largest problem sizes, enabling computations that would otherwise be memory-constrained.

The results establish a clear performance progression based on problem scale and implementation strategy. While basic row-wise decomposition provides reasonable speedups, our optimized block-wise approach demonstrates substantially better scaling characteristics. The investigation also highlights important trade-offs in communication strategies - while extended halo regions reduce synchronization frequency, they introduce memory bandwidth limitations in shared-memory configurations. Future work should explore hybrid approaches combining block-wise decomposition with multi-node execution, as well as investigate asynchronous communication patterns to further optimize parallel efficiency.