HPC - 3. Assignment - 2D Gray-Scott Model in CUDA

Kristjan Kostanjšek, Nejc Ločičnik

I. Introduction

The Gray-Scott model represents a fundamental reaction-diffusion system that simulates pattern formation through the spatial interaction of chemical concentrations. This computationally intensive model is particularly well-suited for GPU acceleration, as each grid cell's evolution depends only on its local neighborhood, enabling massive parallelism. Figure 1 demonstrates the characteristic Turing patterns our implementation produces, showcasing the model's ability to generate complex structures from simple rules.

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

Our implementation systematically evaluates three parallelization strategies, achieving remarkable speedups particularly for large grid sizes. The optimal multi-GPU configuration demonstrates up to 1677× acceleration for 4096×4096 grids, with even greater benefits (2× faster than single-GPU versions) when scaling to 8192×8192 resolutions. These results highlight how careful architecture-aware optimization can unlock the full potential of modern GPU hardware for scientific simulation.

II. Optimizations and Results

The following subsections describe the optimizations applied to the sequential code and the computational speed-up achieved for each optimization. Starting from the initial sequential implementation, we performed three key optimizations: (1) basic GPU acceleration, (2) GPU optimization with shared memory, and (3) multi-GPU optimization, where the workload was distributed across two GPUs.

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. The sequential timings represent an average of three runs, while all GPU timings are averaged over ten runs for statistical reliability.

In this section, we present the performance results of the baseline program and its optimized variants, compare their execution times, and provide a detailed analysis of the findings.

A. Sequential Version

We first developed a sequential version of the program and executed it across five different grid sizes, with each simulation limited to 5,000 steps. The measured execution times (in seconds), corresponding to each grid size, are presented in Table I. Notably, these timings reflect only the Gray-Scott simulation itself, excluding grid initialization to ensure accurate performance evaluation.

grid size	256x256	512x512	1024x1024	2048x2048	4096x4096
$t_S[s]$	12.74	46.44	182.44	723.05	2878.27

Table I
Execution times for sequential program.

B. Basic GPU Parallelization

The Gray-Scott simulation is inherently parallelizable, as each grid cell's computation remains independent within a single simulation step. Leveraging CUDA, we assigned each cell to an individual thread, enabling full-grid parallel processing per step. Optimal performance was achieved using a 32×32 thread block configuration, maximizing GPU occupancy.

Table II summarizes the execution times and speedups achieved across varying grid sizes. As with the sequential benchmarks, timing focuses solely on the simulation kernel, excluding grid initialization and host-device data transfers.

grid size	256x256	512x512	1024x1024	2048x2048	4096x4096
$t_P[s]$	0.247	0.278	0.251	0.557	1.892
t_S/t_P	51.58	167.05	726.85	1298.11	1521.28

Table II
Execution times and speedups for basic GPU parallel optimization.

The results demonstrate dramatic speedups, particularly for larger grids. While small grids (e.g., 256×256) already show a 51.58× improvement, the benefit scales nearly linearly with problem size, reaching 1521× acceleration for the 4096×4096 case. This highlights the GPU's efficiency in handling data-parallel workloads, where computational overhead is amortized over massive parallelism.

C. GPU Shared Memory Utilization

While the basic GPU implementation processes each cell directly from global memory, further optimization can be achieved by leveraging shared memory. In this approach, each thread block loads its working tile into shared memory, reducing global memory accesses. Boundary cells are also loaded to accommodate the stencil pattern, introducing some overhead. A 32×32 block size was maintained to balance shared memory usage and boundary overhead.

Table III presents the execution times and speedups for this optimization. As with previous tests, timings exclude grid initialization and host-device transfers.

grid size	256x256	512x512	1024x1024	2048x2048	4096x4096
$t_P[s]$	0.210	0.182	0.268	0.580	2.383
t_S/t_P	60.66	255.16	680.74	1246.64	1207.83

Table III
Execution times and speedups for GPU shared memory utilization.

The results reveal mixed performance gains. Smaller grids (256×256 to 1024×1024) show modest improvements (up to 255× speedup), but performance degrades for larger

grids—most notably for the 4096×4096 case, where shared memory introduces a 26% slowdown compared to the basic GPU version. We attribute this to the Gray-Scott model's simple stencil pattern and low per-thread computation, which allows the GPU's L1/L2 caches to effectively mask global memory latency. In such cases, shared memory's overhead—particularly from redundant boundary loading—outweighs its benefits.

D. Mulitple GPUs Utilization

Our final optimization implemented a multi-GPU solution using Open MPI, distributing the computational workload across two GPUs through horizontal grid partitioning. This approach maintained optimal memory access patterns while requiring periodic boundary data exchange between devices via MPI messages. The communication overhead presents opportunities for further optimization through asynchronous computation during data transfers.

grid size	256x256	512x512	1024x1024	2048x2048	4096x4096
$t_P[s]$	0.636	0.777	0.921	1.198	1.923
t_S/t_P	20.03	59.77	198.09	603.55	1496.76

Table IV
Execution times and speedups for multiple GPU (2) utilization.

The performance results in Table IV reveal distinct scaling characteristics. Smaller grids (256×256 to 1024×1024) show significant MPI overhead, yielding more modest speedups of 20.03× to 198.09× compared to single-GPU implementations. However, the benefits become pronounced at larger scales, with the 4096×4096 grid achieving a 1496.76× speedup. The additional comparison in Table V demonstrates the multi-GPU approach's superiority for extreme problem sizes - at 8192×8192 resolution it runs in 4.68s versus 7.58s (basic) and 9.24s (shared memory), and successfully processes a massive 32768×32768 grid in 63.50s where single-GPU implementations fail due to memory constraints.

grid size	8192x8192	16384x16384	32768x32768
basic $t_P[s]$	7.58	26.04	NO MEMORY
shared $t_P[s]$	9.24	33.10	NO MEMORY
mpi $t_P[s]$	4.68	14.41	63.50

Table V
Execution time comparison for different optimizations on larger grids.

These results clearly demonstrate the trade-off between communication overhead and parallel processing gains. While the MPI implementation shows limited benefits for smaller grids, it becomes essential for handling very large simulations, both in terms of performance and memory capacity. The ability to process grids up to 32768×32768 - impossible with single-GPU configurations - highlights the method's value for large-scale scientific simulations. The consistent 1.6-2× advantage over single-GPU versions at extreme scales confirms that multi-GPU processing becomes increasingly worthwhile as problem size grows beyond single-device capabilities.

III. Conclusion

Our comprehensive optimization of the Gray-Scott reaction-diffusion model has demonstrated the effectiveness of GPU acceleration for scientific simulation. The study reveals three key findings: (1) Basic GPU parallelization delivers excellent scaling (up to 1496× speedup for 4096×4096 grids), (2) Shared memory optimizations provide limited benefits for this particular stencil pattern, and (3) Multi-GPU processing becomes essential for extreme-scale simulations, enabling computations impossible on single devices (successfully handling 32768×32768 grids).

The results highlight a clear performance hierarchy based on problem scale. While single-GPU implementations suffice for moderate grid sizes, our MPI solution shows its true value at scale - processing 8192×8192 grids 1.6-2× faster than single-GPU versions and supporting simulations that would otherwise fail due to memory constraints. The communication overhead (20-198× speedup for smaller grids vs 603-1496× for larger ones) presents a clear trade-off that future work should address through asynchronous communication patterns and generalized MPI support for arbitrary GPU counts.