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
speedup	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
speedup	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

For our final optimization, we implemented a multi-GPU solution using Open MPI to distribute the computational workload across two GPUs. The grid was split horizontally (for optimal memory access patterns) with each GPU processing half of the domain. Boundary data between partitions was exchanged using MPI messages, which introduced some communication overhead (though this could potentially be further optimized by having the GPU work, while messages are being exchanged).

Table IV shows the execution times and speedups achieved with this multi-GPU approach. Consistent with previous measurements, these timings include only the simulation kernel execution, excluding initialization and data transfer overheads.

grid size	256x256	512x512	1024x1024	2048x2048	4096x4096
$t_P[s]$	0.609	0.580	0.548	0.795	1.716
speedup	20.92	80.07	332.92	909.49	1677.31

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

The results demonstrate an interesting scaling pattern. For smaller grids (256×256 to 1024×1024), MPI communication overhead significantly impacts performance, resulting in speedups substantially lower than the single-GPU implementations (20.92× to 332.92×). However, as grid size increases, the benefits of parallel processing outweigh the communication costs - we observe a 1677× speedup for the 4096×4096 case. Additional testing with an 8192×8192 grid revealed even more pronounced advantages, with the multi-GPU implementation (≈4s) outperforming both the basic (≈7s) and shared memory (≈8-9s) single-GPU versions. This confirms that while MPI introduces overhead, it becomes increasingly beneficial for very large problem sizes where computational workload dominates communication costs.

III. Conclusion

In this study, we implemented and optimized the Gray-Scott reaction-diffusion model using CUDA, with particular focus on the stencil computation stage which forms the core computational workload. We explored the trade-offs between shared memory optimizations and communication overhead in our multi-GPU implementation, demonstrating how different approaches suit different problem scales. Our parallel implementations achieved remarkable speedups up to 1677× for 4096×4096 grids, with the multi-GPU configuration proving particularly effective for very large simulations (8192×8198), where it outperformed single-GPU versions by 2×.

Future work could investigate asynchronous communication patterns to overlap computation and MPI transfers, potentially reducing the overhead observed in medium-sized grids. A key improvement would be to generalize the MPI code to handle an arbitrary number of GPUs, as the current implementation is essentially hardcoded for two devices. These improvements would make the simulation more flexible and scalable for large-scale grids that potentially don't even fit on a single GPU.