Advection Simulation

Ya Wei Tsai
CNET ID: jeremyyawei

January, 2025

Decomposition Strategy

The decomposition strategy used in this project is **manual Cartesian grid decomposition**. Instead of relying on external libraries or built-in MPI functions for domain partitioning, the grid is explicitly divided into subdomains based on user-specified dimensions (`dim_0` and `dim_1`). Each MPI rank is manually assigned a portion of the grid.

The number of rows and columns in each subdomain (`Nx` and `Ny`) is calculated using basic arithmetic:

- Each subdomain receives a base number of rows and columns.
- Remainders from uneven divisions of the grid are distributed across the first few ranks, ensuring balanced workloads.

Ghost cell communication is also manually implemented by explicitly identifying each rank's neighbors (top, bottom, left, right) based on its position in the grid.

Ghost Cell Exchange

To handle boundary conditions between subdomains, **ghost cells** were implemented. Ghost cells store overlapping boundary data exchanged between neighboring subdomains. The `MPI_Sendrecv` function was used to exchange ghost cell data between neighboring MPI ranks. This approach ensured consistency at subdomain boundaries while maintaining efficient communication. For larger ghost layers, data exchange was carefully optimized to minimize communication overhead.

Parallel Output

Parallel output was implemented by enabling each MPI rank to independently write its subdomain data to binary files. This decentralized approach reduced the bottleneck associated with centralized file output and improved scalability. The outputs were named according to their MPI rank and simulation step, facilitating reconstruction of global data for visualization and analysis.

Parallelization with MPI and OpenMP

The project utilized both **MPI** and **OpenMP** for hybrid parallelization:

- **MPI:** Used for inter-node parallelization, where each MPI rank was assigned a subdomain and responsible for communication with neighboring ranks.
- **OpenMP:** Used for intra-node parallelization, leveraging multiple threads within each node to process subdomain computations in parallel. Each MPI rank spawns multiple threads to perform computations efficiently.

This hybrid approach maximized the use of available hardware, ensuring scalability across diverse configurations. The project also included performance measurements, such as execution time and grind rate, to evaluate and compare the efficiency of different configurations.

Verification

Carry out the verification tests described below using the values specified in Table 2. For each simulation, show solutions at the specified times ($T = 0.0$, $t = 0.5T$, and $t = T$) using colormaps or Z-axis plots:

- **Serial Lax,** $N = 4000$

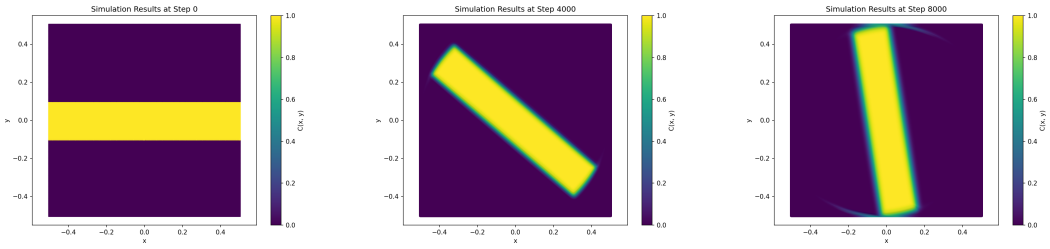

Figure 1: Serial Lax: Solutions at $t = 0.0$, $t = 0.5T$, and $t = T$.

- **Shared Memory Parallel Lax,** $N = 4000$**,** $ncores = 16$

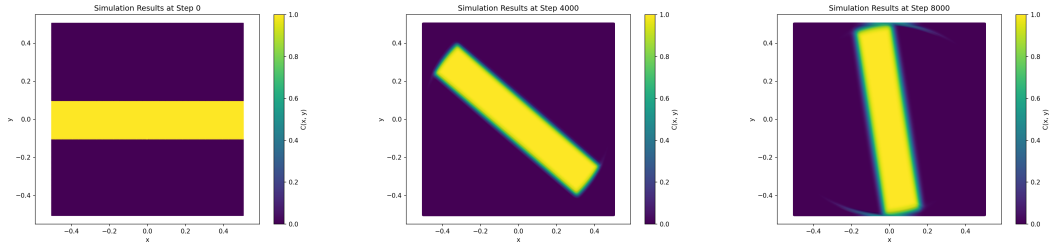

Figure 2: Shared Memory Parallel Lax: Solutions at $t = 0.0$, $t = 0.5T$, and $t = T$.

- **Distributed Memory Parallel Lax,** $N = 4000$**, 1 core per node, 4 nodes**

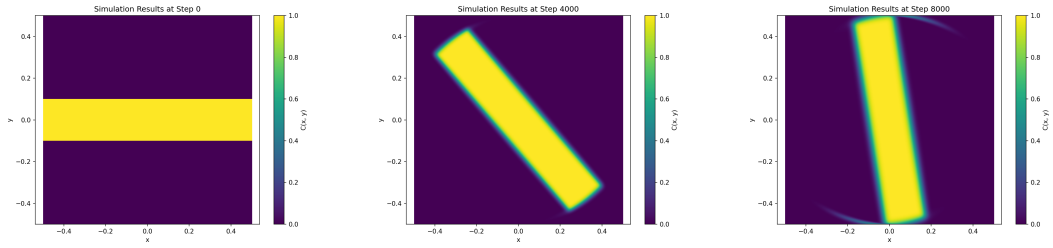

Figure 3: Distributed Memory Parallel Lax: Solutions at $t = 0.0$, $t = 0.5T$, and $t = T$.

- **Hybrid Lax,** $N = 4000$**, 16 cores per node, 16 nodes, 1 MPI rank per node**

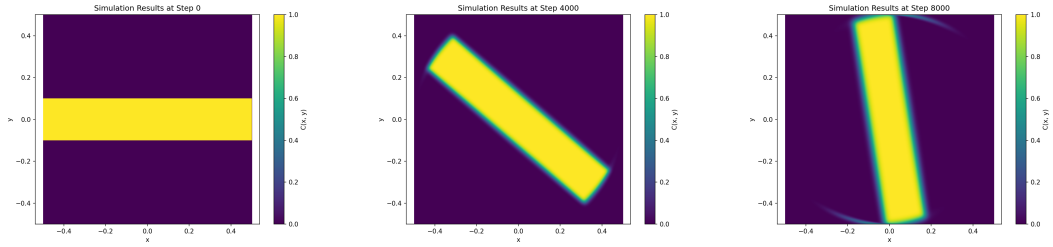

Figure 4: Hybrid Lax: Solutions at $t = 0.0$, $t = 0.5T$, and $t = T$.

Performance

Record the performance data by timing only the outer loop over timesteps (excluding I/O and initialization). State your best runtime and the associated configuration.

Note: Requirements for MPI/OpenMP Solver

- Use a square domain with square decomposition as explained in the supplemental video.
- `C` and `Cnew` data structures must be decomposed across MPI ranks.
- Input parameters should be read on rank 0 and broadcast to other ranks.
- Choose any parallel I/O strategy.

Best Runtime and Associated Configuration

Programming Model	Nodes	Cores Per Node	Execution Time (sec)	Grind Rate
Hybrid: One MPI rank per node	3x3	16	6.41	19,964,895,442.50

Table 1: Best performance configuration.

Parameter Values

Parameter	Value
L	1.0
Domain	$[-L/2, L/2]$
t_{final}	1.0
u	$\sqrt{2}y$
v	$-\sqrt{2}x$
dt	Any stable value
N	4000
dx	$L/(N-1)$
Boundary condition	Periodic
Initial condition	$C([-L/2, L/2], [-0.1, 0.1]) = 1.0$

Table 2: Parameter values for verification tests.

Programming Model	Nodes	Cores Per Node	Execution Time (sec)	Grind Rate
MPI-Only	1	1	361.33	354,289,330.39
MPI-Only	2x1	1	184.86	692,489,575.04
MPI-Only	2x2	1	93.51	1,369,071,264.24
MPI-Only	3x3	1	43.13	2,968,284,873.07
MPI-Only	4x4	1	29.17	4,389,046,274.66
OpenMP-Only	1	1	355.77	359,829,482.94
OpenMP-Only	1	2	211.87	604,219,453.34
OpenMP-Only	1	4	89.68	1,427,528,988.22
OpenMP-Only	1	9	52.58	2,434,860,425.59
OpenMP-Only	1	16	59.43	2,154,018,167.40
Hybrid: One MPI rank per node	1	16	39.15	3,269,492,379.66
Hybrid: One MPI rank per node	2x1	16	26.91	4,757,872,688.61
Hybrid: One MPI rank per node	2x2	16	14.24	8,989,649,979.73
Hybrid: One MPI rank per node	3x3	16	6.41	19,964,895,442.50
Hybrid: One MPI rank per node	4x4	16	7.54	16,979,742,410.21

Table 3: Performance test configurations and execution times.

Analysis

Figure 5 shows the execution time for various configurations. The key observations are as follows:

- **MPI-Only Model:** Increasing the number of nodes significantly reduces execution time due to improved parallelism. However, communication overhead between nodes prevents it from being the most efficient.
- **OpenMP-Only Model:** Execution time decreases as the number of cores increases. It performs well for single-node configurations but lacks the scalability of distributed memory models.
- **Hybrid Model:** The Hybrid model combines MPI and OpenMP, achieving the best balance of parallelism and minimal communication overhead. The configuration with **3x3 nodes and 16 cores per node** achieves the fastest execution time of **6.41 seconds**, with the highest grind rate of **19,964,895,442.50 points per second**.

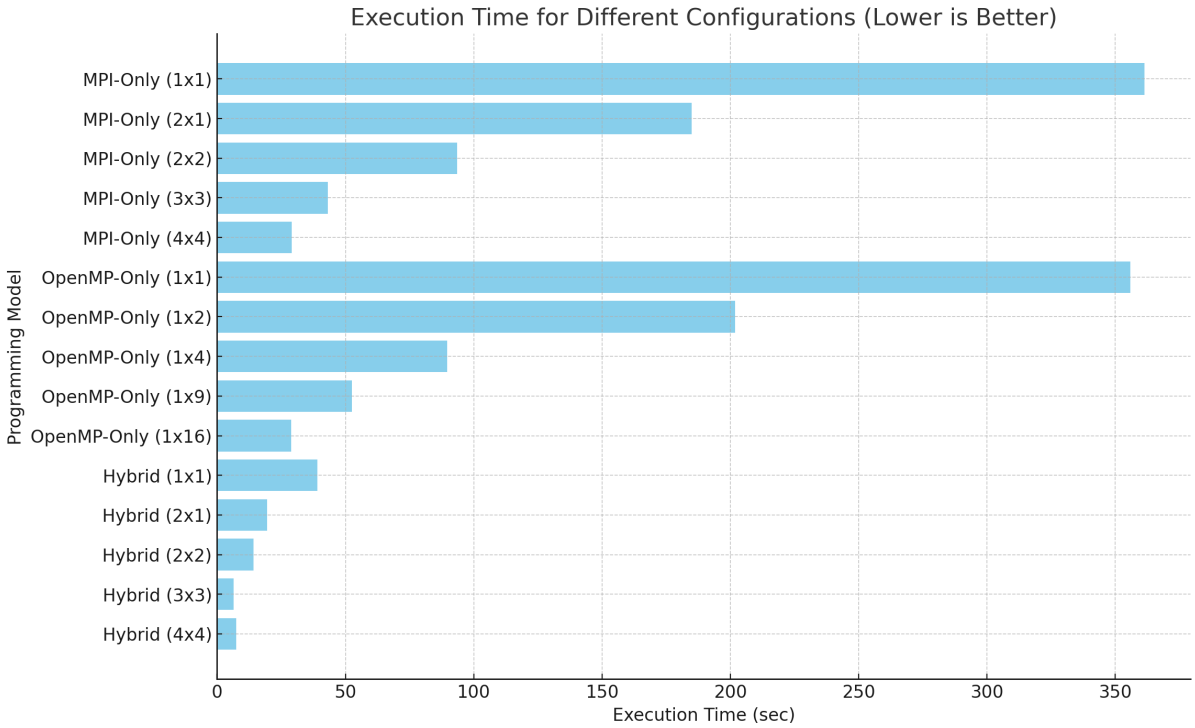

Figure 5: Execution time for different configurations and programming models.