Milestone 06

Andreas Greiner, Lars Pastewka

Table of contents

Distributed memory parallelization with MPI

Learning goals

The student will…

- …understand the fundamentals of distributed-memory parallelization using MPI.
- …be able to decompose a domain across multiple processes.
- …be able to exchange data between processes using point-to-point and collective communication.
- …understand ghost cells and halo exchange patterns.

Introduction

In the previous milestones, you implemented a Lattice Boltzmann solver that runs on a single CPU core or GPU using Kokkos for parallelization. While Kokkos allows you to effectively parallelize computations across available processors within a single node, many practical applications require simulations larger than what fits in the memory of a single machine or benefit from the combined computational power of multiple nodes in a high-performance computing (HPC) cluster.

Distributed-memory parallelization using the Message Passing Interface (MPI) allows us to solve this problem by distributing the computational domain across multiple processes, each potentially running on different nodes. Each process owns a portion of the domain and communicates with neighboring processes to exchange information at domain boundaries.

The primary reading for this milestone is the domain decomposition lecture, which covers ghost cells, halo exchange, which populations must be communicated, corner handling and the strip-vs-tile trade-off. The MPI lecture covers the underlying communication primitives and the integration with Kokkos.

Concept: Domain Decomposition

The key idea of domain decomposition is to divide the simulation domain into subdomains, with each MPI process handling one subdomain (see Figure 1). In your 2D Lattice Boltzmann simulation, you might decompose the domain in the x-direction, the y-direction, or in both dimensions.

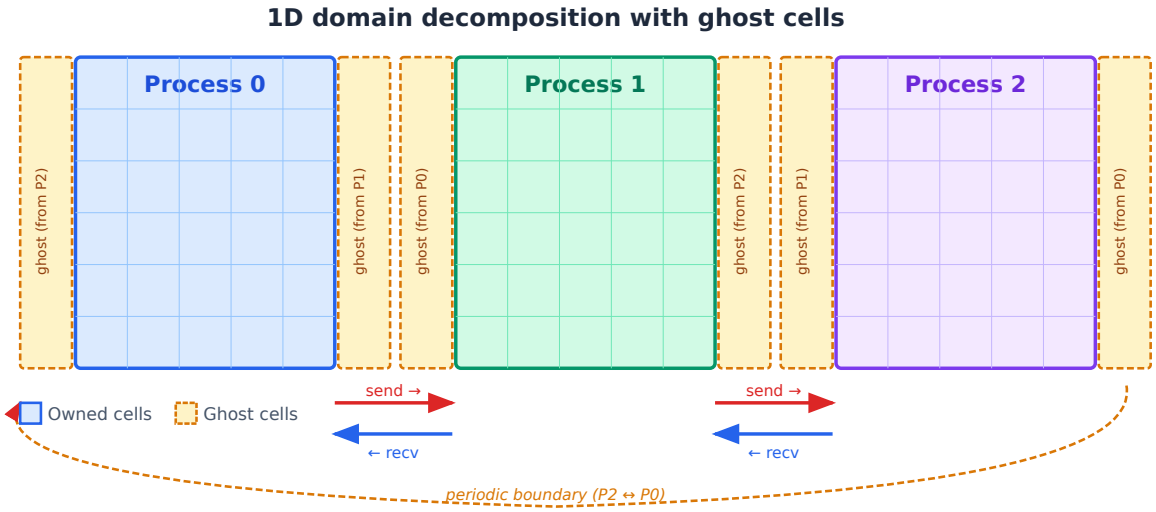

Figure 1: Domain decomposition: the simulation domain is split into subdomains, each owned by one MPI process. Each subdomain is padded with a layer of ghost (halo) cells that mirror the boundary cells of the neighboring processes and are filled by communication.

For the streaming step in the Lattice Boltzmann method, neighboring grid points exchange data (the f_i values). When a process computes the streaming for its boundary grid points, it needs values from grid points in neighboring subdomains. These are handled via **ghost cells** or **halo cells** – extra rows or columns that are populated with data from neighboring processes.

A typical workflow for each time step is:

1. **Compute locally**: Each process performs streaming and collision operations on its interior grid points.
2. **Exchange ghost cells**: Send and receive boundary data from neighboring processes.
3. **Compute boundaries**: Process the boundary grid points using the received ghost cell values.
4. **Synchronize**: Use collective communication to synchronize global quantities (e.g., reduce to compute global density, kinetic energy).

Tasks

- Decompose your 2D domain along one or both spatial dimensions and distribute it across MPI processes.
- Implement a ghost cell / halo exchange pattern for the distribution function f_i. Consider whether you need to exchange ghost cells after streaming, before collision, or both. Think about the minimum amount of data that needs to be communicated.
- Parallelize your streaming and collision operations so that each process works only on its assigned subdomain. Use Kokkos `parallel_for` loops to parallelize within each MPI process.

- Implement collective communication (using `MPI_Allreduce` or similar) to compute global quantities such as:
  - Total mass (sum of all ρ values across all processes)
  - Kinetic energy or other diagnostics needed for validation
- Validate your parallelized code by running it with different numbers of processes and comparing results to your serial implementation. The results should be identical (up to numerical precision).
- **Performance analysis**: Measure the strong scaling of your code by running the same problem on increasing numbers of processes. Create a plot showing speedup vs. number of processes. See the scaling notes for how to set up and plot scaling runs, and the bwUniCluster notes for where to run them.
- *Optional*: Also perform a weak-scaling study, in which the problem size grows proportionally with the number of processes so that the work per process stays constant.

Notes

- The MPI standard provides multiple communication strategies for ghost cell exchange. The most straightforward approach is to send boundary data in the positive x-direction to the right neighbor and receive from the left neighbor, and similarly for the y-direction. Alternatively, you can use `MPI_Sendrecv` to do send-receive pairs in a single call, which helps avoid deadlock issues.
- Be mindful of the data type you send. Consider whether you need to send the full f_i for all 9 velocities or a subset, depending on your chosen domain decomposition strategy. Only the populations pointing *across* a boundary actually need to be communicated — e.g. for a right boundary these are channels 1, 5 and 8, as illustrated in Figure 2 (see the channel table in the boundary conditions lecture and the discussion in the domain decomposition lecture) — although sending all 9 is simpler and also correct.
- On an HPC cluster, use the batch scheduler (e.g., SLURM) to submit and run your MPI jobs. See the bwUniCluster notes and running MPI programs for guidance.
- If you use GPU acceleration (CUDA or HIP), consider using GPU-aware MPI if your cluster supports it. Refer to the Kokkos GPU compilation notes for details.
- Document your communication pattern (which processes exchange which data) and the ghost cell width for your implementation.
- Present a strong scaling plot demonstrating the speedup with the number of processes. Discuss the efficiency and any bottlenecks observed.

Suggested approach

1. Start with 1D domain decomposition (decompose only in the x-direction, for example).
2. Verify correctness by comparing results with your serial implementation for a small problem on a few processes.
3. Extend to 2D domain decomposition if time permits.
4. Conduct scaling studies to understand the parallel efficiency of your implementation.

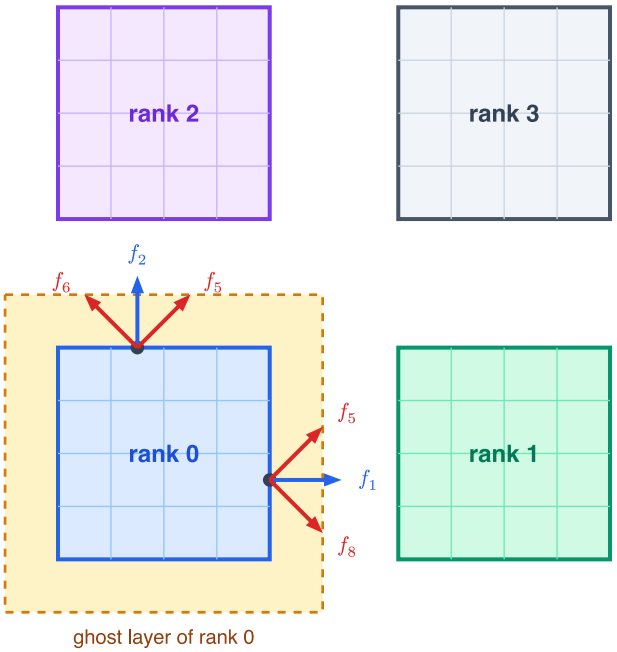

Figure 2: Only populations pointing across a subdomain boundary must be communicated: channels 1, 5 and 8 cross a right boundary, channels 2, 5 and 6 a top boundary. The ghost layer (dashed) receives the incoming populations.