Domain decomposition in Eiron (and EIRENE?)

Oskar Lappi

November 2024

Simulation state and context

I will explain in broad strokes how domain decomposition is implemented in Eiron and what requirements/features are important.

After this I'll hand out a sheet with the same information in the form of an exercise that we can all try solving together.

NOTE: This way will not be the only possible way to do domain decomposition, but it's one way that has now been implemented, and therefore has some guarantees of actually working.

Simulation state and context

Eiron uses two main data structures when simulating:

- Particle state
    - Particle position, velocity, Monte Carlo parameters, etc.
    - Last particle event that occured
    - Particle source id (stratum id in EIRENE?)
    - Particle id
    - Position in RNG stream (how many random numbers has the particle used)
- Simulation context
    - Grids for the current species and subdomains
    -
        - Collision grid
        - Estimation grid if estimating (tally grid)
    - Simulation parameters
    - Collision processes
    - Particle sources

The particle state is what is sent between MPI ranks, the simulation context consists of resources that are local to an MPI rank.

Simulation API

This let's us design a simulation API that looks like this

- simulate_step(state, context) -> state

Simulation loop

We can now sketch the simulating loop of a domain decomposition program:

1. Receive particle in a subdomain that's in the set of local simulation contexts
2. Check the particle species and position, select the subdomain resources at that location for that species
3. Run simulate_step
4. Check the resulting state from simulate_step, if the particle event it returns is of type SUBDOMAIN_BOUNDARY_CROSSED, we send the particle forward to a process that has that subdomain
5. Process the next particle

Generation loop

1. If the system needs more active particles
2. Generate a particle from a source
3. Send it to a process that has the corresponding subdomain in its simulation context

Full program

1. Check if generation is necessary
2. If yes, generate a particle, send it forward, and go to 1
3. If no, go to 4
4. Try to receive a particle
5. If a particle was received, simulate it
6. If the simulated particle crossed a subdomain, send it forward, and go to 4
7. If no particle was received go to 1

We also need to keep track of how many particles we've generated, and count that each one has terminated.

Particle termination

The simplest way to make sure that all particles are simulated to the end is to have one MPI rank responsible for generating particles from each source, and sending a message to that MPI rank whenever a particle from that source has terminated.

The generator will count the number of incoming particle termination messages, and broadcast a message to all MPI ranks after that saying this source/stratum is completely simulated.

If there are N generators, then all MPI ranks must wait for N such simulation-complete messages. Only then can all MPI ranks exit the main program loop.

Grid splitting and joining

We will need a way to split grids into subdomains, and to join them together at the end (or we can use a distributed file format like HDF5 to write the subdomains separately).

Eventually, when scaling up, splitting the domain will not be possible, because it won't fit into memory, and then we will have to initialize the grids in a distributed way. But this is an advanced feature, probably not so relevant to begin with.

RNG

We also need to rethink random numbers. Each particle has to have its own random number seed, which seeds a RNG, and keep track of how many random numbers have been generated for it.

Whenever a new process receives a particle and starts simulating it, it has to seed a RNG and skip the correct amount of random numbers to get to the correct position in the RNG stream again.

If we don't do this, simulations will not be deterministic, and may also suffer from bad statistics.

Exercise

This session is supposed to be a discussion, so I'd like you to grab an exercise paper which contains all these requirements, read it through, and then we can start discussing them together.

NOTE: This way will not be the only possible way to do domain decomposition, but it's one way that has now been implemented, and therefore has some guarantees of actually working.