Parallel Design Patterns CW1

Report on Parallelization of the Brain Simulation Code

B185276

February 2025

Contents

Introduction

The medical research company aims to simulate signal propagation in the human brain by representing neurons and nerves as a large graph. The existing serial code models multiple signal types and random stimulation events but struggles with scalability when the network grows to thousands or even millions of neurons. This limitation hinders the ability to run longer or more detailed simulations. Consequently, the primary challenge is to parallelize the code so it can efficiently handle large graphs without compromising the accuracy of signal interactions among neurons.

1 Question 1

1.1 (a)

Two parallel patterns from the design space that are well-suited to this brain simulation problem are Geometric Decomposition and the Actor Pattern.

Geometric Decomposition

Geometric Decomposition partitions the neuron network into smaller subdomains, each assigned to a separate execution unit. The main goals are to maintain data locality, reduce communication to boundary interfaces, and balance workloads across subdomains. Key forces include minimizing inter-domain exchanges, ensuring an even distribution of computational effort, and preserving a straightforward iterative structure.

Advantages

- High data locality, reducing remote communication.
- Well-established domain-decomposition tools that handle subdomain boundaries.
- Scales effectively on HPC systems if subdomains are balanced.

Disadvantages

- Non-trivial partitioning if neurons have irregular connectivity.
- Boundary exchanges add overhead and complexity of implementation.
- Potential load imbalance when neuron activity is unevenly distributed.

Suitability Geometric Decomposition aligns with typical HPC practices for large-scale simulations. By clustering neurons that are spatially or topologically close, most updates occur locally, and only boundary neurons require synchronization. This approach can deliver good parallel performance if partitions are chosen to minimize inter-domain communication.

Actor Pattern

The Actor Pattern treats each neuron as an independent actor that communicates through message passing. It naturally captures the event-driven behavior of neural firing, with each actor maintaining its own state. The main forces include isolating the state to prevent data races, managing high volumes of messages among neurons, and adapting to unpredictable firing rates without centralized control.

Advantages

- Clear mapping between neuron signals and asynchronous messages.
- Encapsulation of the neuron state, reducing concurrency conflicts.
- Flexible handling of dynamic firing patterns and heterogeneous neuron behavior.

Disadvantages

- Large messaging overhead if the network is densely connected.
- Load balancing is less direct, as firing rates vary over time.
- Debugging and ensuring global coordination can be more complex.

Suitability The Actor Pattern suits neural models with frequent, asynchronous events. The firing of each neuron can be represented by messages sent to connected actors, and the local state is updated independently. This approach is effective in capturing irregular workload patterns but requires a robust messaging layer to avoid bottlenecks.

1.2 (b)

Geometric Decomposition is selected for this case due to several reasons:

- Uses Established HPC Tools: Many scientific codes rely on domain-based partitioning libraries.
- Simplifies Iteration: Each subdomain updates its local neurons and then synchronizes only at the boundaries.
- Manages Communication Overhead: Most data movement remains within each subdomain, which is easier to optimize than frequent small messages.

The Actor Pattern is more flexible for irregular firing, but it can lead to high messaging overhead and more complex coordination. Geometric Decomposition provides a clearer route to large-scale performance and is simpler to integrate with existing HPC frameworks.

1.3 (c)

Geometric Decomposition often involves domain-partitioning libraries (e.g., METIS or PT-Scotch) and communication protocols like `MPI`. C/C++ with MPI or Fortran with MPI are common choices in HPC environments. Distributed memory platforms (such as clusters) are well-suited because each process handles a subdomain and exchanges boundary data. If GPUs are used, frameworks like CUDA or OpenACC can handle local computation, with subdomain transfers relying on message-passing between devices.

1.4 (d)

With Geometric Decomposition, load imbalance may occur if neurons or connections are unevenly distributed. Boundary communication can also become a bottleneck if many signals cross partitions. Possible ways to address these issues include:

- Partition Quality vs. Overhead: A more precise partition reduces communication but increases the complexity of partitioning steps.
- Static vs. Dynamic Decomposition: Static partitions are simpler but may become unbalanced over time, while dynamic methods adjust workloads but add runtime overhead.
- Subdomain Size vs. Parallelism: Smaller subdomains can increase parallelism but also increase boundary exchanges.

Balancing these factors is essential for achieving good parallel performance and scalability.

2 Question 2

Two patterns are identified as unsuitable for this brain simulation problem.

Pipeline

A pipeline pattern arranges tasks into a series of stages, where each stage processes data and passes the result on to the next. This works well when data flows in one direction and each stage performs a stable, well-defined operation. However, the brain simulation differs in several ways:

- Neurons fire in unpredictable bursts and can send signals back to earlier neurons, creating loops rather than a one-way flow.
- A pipeline typically requires each stage to handle roughly the same workload, but neuron activity can vary significantly over time and across the network.
- Signals may merge, split, or converge from multiple neurons, which goes against the straightforward stage-to-stage approach.

Due to these feedback loops, fluctuating workloads, and the bidirectional nature of signals, the pipeline structure does not fit the neural model.

Divide and Conquer

Divide and Conquer splits a problem into smaller subproblems, solves them recursively, and then merges the results. This pattern suits tasks with clear divisions and predictable merge steps (e.g., sorting or matrix multiplication). However, the brain simulation differs in several ways:

- Neurons may connect to distant regions of the brain, meaning subproblems cannot be isolated as neatly as in typical recursive divisions.
- After solving partial steps, combining updated neuron states is non-trivial when signals traverse multiple paths and timing influences outcomes.
- The simulation includes random firing and complex feedback, which undermines the straightforward "divide, solve, merge" cycle.

Due to these reasons, Divide and Conquer lacks a natural way to decompose a neural network and then reassemble the partial solutions reliably. As a result, implementing it would be significantly more complex and less efficient than other patterns.

3 Question 3

3.1 (a)

- Profile Execution: Tools such as `gprof`, `perf`, or Intel VTune can identify hotspots, measure function-level performance, and locate bottlenecks in the serial code.
- Instrument Key Sections: Timers or counters can be inserted around critical functions to determine how frequently they are called and how long they take, indicating where optimization is most needed.
- Analyze Memory Usage: Cache misses, memory bandwidth, and data-access patterns can be tracked to determine if memory layouts or access strategies are causing performance issues.
- Compare with Test Cases: Smaller input sets can be used to check correctness, measure run times, and compare results against expected outcomes for performance baselines.

3.2 (b)

- Scaling Tests: The number of cores can be varied while keeping the problem size fixed (strong scaling) or increasing it (weak scaling) to observe how effectively the code uses additional resources.
- Parallel Profilers: Profilers such as `mpiP`, `Scalasca`, or `HPCToolkit` can measure communication overhead, function call times, and load imbalance across nodes or threads.
- Speedup and Efficiency: Parallel run times can be compared against the serial baseline to calculate speedup and determine efficiency, highlighting how effectively each added core reduces execution time.

- Roofline Analysis: The balance between computational throughput and memory bandwidth can be examined by plotting performance against theoretical hardware limits to see which factor constrains scaling.

4 Conclusion

In this report, methods were explored for parallelizing a brain simulation that models neurons as nodes in a large graph. Two patterns were compared Geometric Decomposition and the Actor Pattern and Geometric Decomposition was selected for its established HPC support and clear partitioning approach. Pipeline and Divide and Conquer were then discussed as unsuitable due to the simulation's irregular signals and bidirectional interactions. Finally, approaches for profiling and scaling the code were described, emphasizing proper domain partitioning and efficient communication. This strategy aims to ensure both high performance and scalability on large HPC systems.