HW4: Genetic Search with MPI - Final Report

Introduction

This report summarizes the implementation and results of a genetic algorithm approach to solve the Reverse Conway's Game of Life problem using MPI for parallelization. The assignment involved several stages of implementation, from serial benchmarking to increasingly sophisticated parallel approaches.

Serial Benchmark (Part 1)

The serial implementation of the Reverse Conway's Game of Life was benchmarked using the provided code. The program was run 10 times with different random seeds to establish a baseline performance.

Hardware Environment

- The serial benchmarks were run on the dev-amd20 node
- Each run used a single CPU core

Results

- **Best Fitness Value**: 274
- **Average Run Time**: 955 seconds

The serial implementation provides a baseline for comparison with the parallel approaches.

Pleasantly Parallel Benchmark (Part 2)

In this stage, the genetic algorithm was run in a pleasantly parallel manner using a job array. Each job ran independently with a different random seed, allowing for exploration of the solution without communication between processes.

Implementation

- Created a job array script (`job_array.sb`) to run 50 independent jobs
- Each job used a different random seed based on the SLURM array task ID
- Each job was allocated 2GB of memory and sufficient runtime based on the serial benchmark

Results

- **Best Fitness Value**: 273

The pleasantly parallel approach allows for exploration of multiple solution paths simultaneously, potentially finding better solutions than the serial approach in the same wall-clock time.

MPI Consolidation (Part 3)

In this stage, MPI was used to consolidate results at the end of the computation. Each worker process ran the genetic algorithm independently, but at the end, they shared their best results with the lead worker (rank 0), which then determined the overall best solution.

Implementation

- Modified `reverseGOL.c` to create `reverseGOL-mpi.c` with MPI support
- Implemented MPI initialization and finalization
- Added code for workers to send their best results to rank 0
- Rank 0 received results from all workers and determined the overall best solution
- Created `mpi_job.sb` to run the MPI program with 50 processes

Results

- **Best Fitness Value**: 273

The MPI consolidation approach maintains the parallel exploration benefits while simplifying the result collection process compared to the job array approach.

Intermittent State Sharing with MPI (Part 4)

In this final stage, the MPI implementation was enhanced to allow workers to share their best results during the computation, not just at the end. This enables the genetic algorithm to benefit from good solutions found by any worker.

Implementation

- Modified `reverseGOL-mpi.c` to implement a ring topology for communication
- After each generation, workers share their best results with their neighbors
- Rank 0 receives from the last worker and sends to rank 1
- Other ranks send to their next neighbor and receive from their previous neighbor
- If a neighbor's solution is better, it replaces the current best solution
- Created `mpi_job.sh` to run the enhanced MPI program

Results

- **Best Fitness Value**: 271

The intermittent state sharing approach allows workers to collaborate during the search process, potentially leading to faster convergence and better solutions.

Conclusion

This assignment demonstrated the power of parallel computing in solving complex optimization problems. By implementing increasingly complex parallel approaches, we were able to explore the solution space more effectively and potentially find better solutions than would be possible with a serial approach.

Compilation Instructions

```
make clean
make
make test
```

Running the Programs

1. Serial benchmark:

   ```
   ./timecheck.sh cmse2.txt
   ```

2. Pleasantly parallel benchmark:

   ```
   sbatch job_array.sb
   ```

3. MPI consolidation:

    ```
    sbatch mpi_job.sb
    ```

4. MPI state sharing:

    ```
    sbatch mpi_job.sh
    ```

Key Files

- `reverseGOL.c` : Original serial implementation
- `reverseGOL-mpi.c` : MPI implementation with state sharing
- `timecheck.sh` : Script for serial benchmarking
- `job_array.sb` : SLURM script for pleasantly parallel execution
- `mpi_job.sb` : SLURM script for MPI consolidation
- `mpi_job.sh` : SLURM script for MPI state sharing
- `serial_best.txt` : Best solution from serial runs
- `pp_best.txt` : Best solution from pleasantly parallel runs
- `mpi_basic_best.txt` : Best solution from MPI consolidation
- `mpi_best.txt` : Best solution from MPI state sharing