Final Exam Part 2 – CSC 410 Parallel Computing

Fall 2024

Due: Dec 17 @ 11:59pm

Given the sequential code for the N-body problem (nBody.c), complete the following tasks.

Coding Task 1: Convert it to parallel using Pthread.

Coding Task 2: Convert it to parallel using OpenMP.

Coding Task 3: Convert it to parallel using MPI.

Coding Task 4: Convert it to parallel using OpenCL. Use the starter code ocl_nbody.c

Task 5: Benchmark the time for the sequential version and parallel versions - coding tasks 1 to 4.

> **Note:** Use ***time ./a.out*** for the sequential version and the appropriate timing function for each parallel programming tool.

Task 6: Explain the following in detail:

a) How did you partition the work with each tool.
b) Explain the necessity of the operations you used in detail for each parallel programming tool.
c) Compare and contrast each tool for parallelizing the N-body sequential program. List their advantages and disadvantages.

Total Points - 60

- Code runs and works as expected - Task 1: 10 points
- Code runs and works as expected - Task 2: 10 points
- Code runs and works as expected - Task 3: 10 points
- Code runs and works as expected - Task 4: 10 points
- Clear documentation - Task 5: 5 points

- Clear and detailed explanation - Task 6: 15 points
- Available on GitHub

Task 6:

For pthreads, the work is divided among pthreads, where the forces for the number of bodies is calculated so the

computation is distributed among each thread. Pthreads are used to handle computation for each subset of bodies.

A barrier is used to make sure that each thread completes their computations for each timestep before heading onto the next one. The Bodies array has shared memory and avoids any race conditions by making sure each threadis operating on it's own subset.

For OpenMP, the work is again divided among threads, but OpenMP takes care of this. OpenMP divides the iterations of

the outer loop among the threads. Atomic operations are used to avoid any race conditions. #pragma openmp parallel

for (static) divides the iterations of the loop among the threads. #pragma omp atomic makes sure that the shared variables

fx and fy do not have any race conditions. Finally, the two separate loops are automatically synchronized at the end of theomp parallel region.

For MPI, the bodies array is again divided among the MPI processes.MPI broadcast makes sure every process has access tothe bodies array. MPI gather is used to collect each updated position and velocity from all processes.MPI_init and MPI_Finalize are called to setup the environment. MPI_Comm_rank and MPI_Comm_size is used to retrievethe process id and and total number of processes

I was unable to get OpenCL to work.Overall, Pthreads has finer controll over each thread, but becomes more complicated when you are explictly managing each thread.

OMP has much simpler syntax which makes writing it a bit easier, and made the inner and outer loop automatically parallel.

There wasn't a whole lot of modification that needed to be done in order to make the sequential version parallel using OpenMP.MPI has the advantage of being able to work with distributed systems, but became more complicated when working with this

specific program. Like OpenMP, the increased flexibility also comes at the cost of increased overhead.

	Sequential	Pthread	OpenMP	MPI	OpenCL
Time (s)	20.436	3.338	18.864	15.158	N/A