The Chinese University of Hong Kong, Shenzhen

Assignment Report

Author:
Muhan Lin

Student Number:
ID 1 1 9 0 1 0 1 7 7

Project 3: N-body Simulation

Muhan Lin 119010177

November 2022

1 Introduction

This project simulates n bodies on a canvas with sequential and parallel implementation respectively. Each body is moving at a velocity with acceleration offered by gravity forces between this body and other ones. If one body collides the canvas boundary or other balls, its velocity in the direction of collision will be reversed. This project implements 5 versions of this program. sequential, mpi, pthread, openmp and cuda. Thread and process synchronization is used to prevent data racing. Additionally, this project compares and analyses the performance of 5 versions with different numbers of bodies and threads or processes. As observed by experiments, when process or thread number is under some threshold, the increasing parallelism can improve efficiency. Among 5 implementations, Openmp and Pthread are the optimal ones, completing tasks with light-weight threads.

2 Code Availability

To compile the sequential implementation without GUI:

```
g++ ./src/sequential.cpp -o seq -O2 -std=c++11
```

To compile the sequential implementation with GUI:

```
g++ ./src/sequential.cpp -o seqg -I/usr/include -L/usr/local/lib -L/usr
    /lib -lglut -lGLU -lGL -lm -DGUI -O2 -std=c++11
```

To compile the MPI implementation without GUI:

```
mpic++ ./src/mpi.cpp -o mpi -std=c++11
```

To compile the MPI implementation with GUI:

```
mpic++ ./src/mpi.cpp -o mpig -I/usr/include -L/usr/local/lib -L/usr/lib
     -lglut -lGLU -lGL -lm -DGUI -std=c++11
```

To compile the pthread implementation without GUI:

```
g++ ./src/pthread.cpp -o pthread -lpthread -O2 -std=c++11
```

To compile the pthread implementation with GUI:

```
g++ ./src/pthread.cpp -o pthreadg -I/usr/include -L/usr/local/lib -L/
    usr/lib -lglut -lGLU -lGL -lm -lpthread -DGUI -O2 -std=c++11
```

To compile the Openmp implementation without GUI:

```
g++ ./src/openmp.cpp -o openmp -fopenmp -O2 -std=c++11
```

To compile the Openmp implementation with GUI:

```
g++ ./src/openmp.cpp -o openmpg -fopenmp -I/usr/include -L/usr/local/
    lib -L/usr/lib -lglut -lGLU -lGL -lm -O2 -DGUI -std=c++11
```

To compile the CUDA implementation without GUI:

```
nvcc ./src/cuda.cu -o cuda -O2 --std=c++11
```

To compile the CUDA implementation with GUI:

```
nvcc ./src/cuda.cu -o cudag -I/usr/include -L/usr/local/lib -L/usr/lib
    -lglut -lGLU -lGL -lm -O2 -DGUI --std=c++11
```

To run the sequential version:

```
./seq $n_body $n_iterations
./seqg $n_body $n_iterations
```

To run the MPI version:

```
mpirun -np $n_processes ./mpi $n_body $n_iterations
mpirun -np $n_processes ./mpig $n_body $n_iterations
```

To run the pthread version:

```
./pthread $n_body $n_iterations $n_threads
./pthreadg $n_body $n_iterations $n_threads
```

To run the Openmp version:

```
openmp $n_body $n_iterations $n_omp_threads
openmpg $n_body $n_iterations $n_omp_threads
```

To run the CUDA version:

```
./cuda $n_body $n_iterations
```

Four sbatch scripts `mpi.sh`, `openmp.sh`, `cuda.sh` and `pthread.sh` are also provided for large-batch experiments conducted for this project on the HPC cluster. You can use or comment the commends provided in these scripts according to your needs. The command to run these scripts is `sbatch xxx.sh`.

3 Methodology

3.1 The Logical Flows

The logical flows of the MPI, pthread, Openmp and CUDA versions are shown as follows.

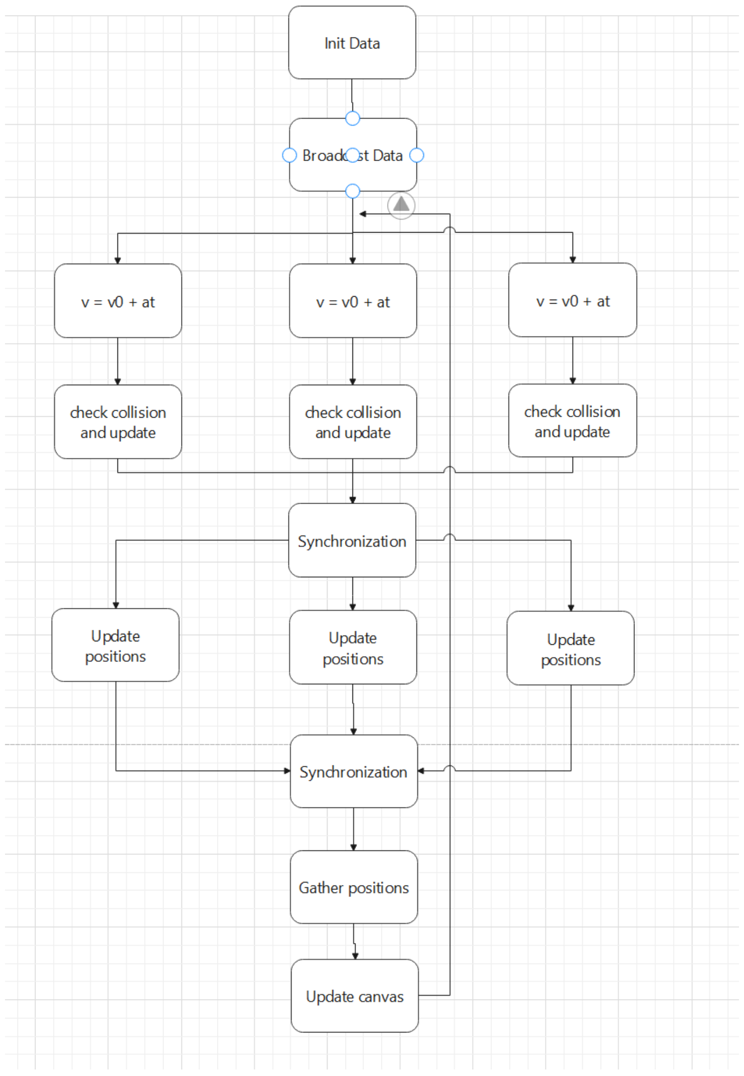

Figure 1: The MPI Flow

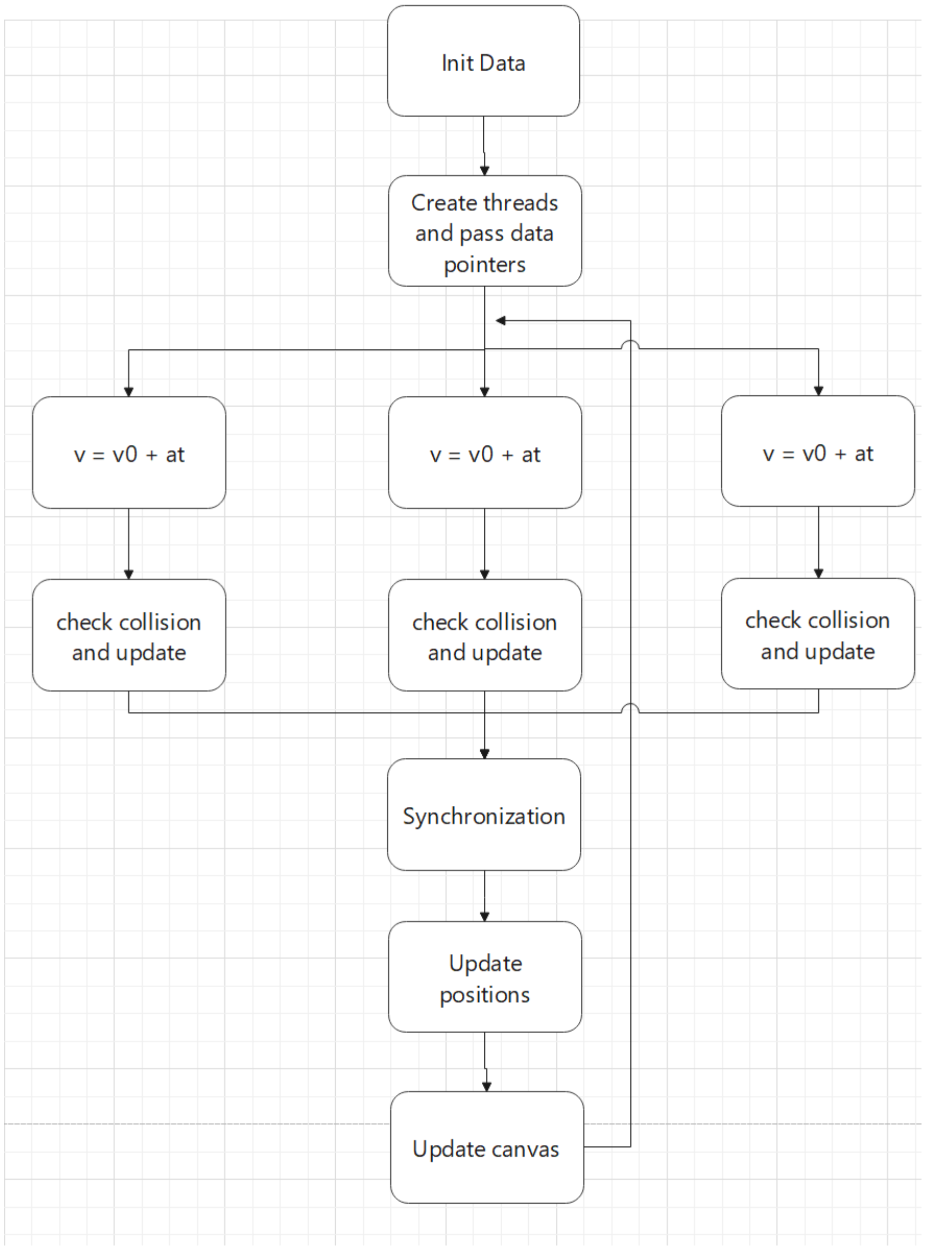
Init Data
Create threads and pass data pointers
v = v0 + at
v = v0 + at
v = v0 + at
check collision and update
check collision and update
check collision and update
Synchronization
Update positions
Update canvas

Figure 2: The pthread Flow

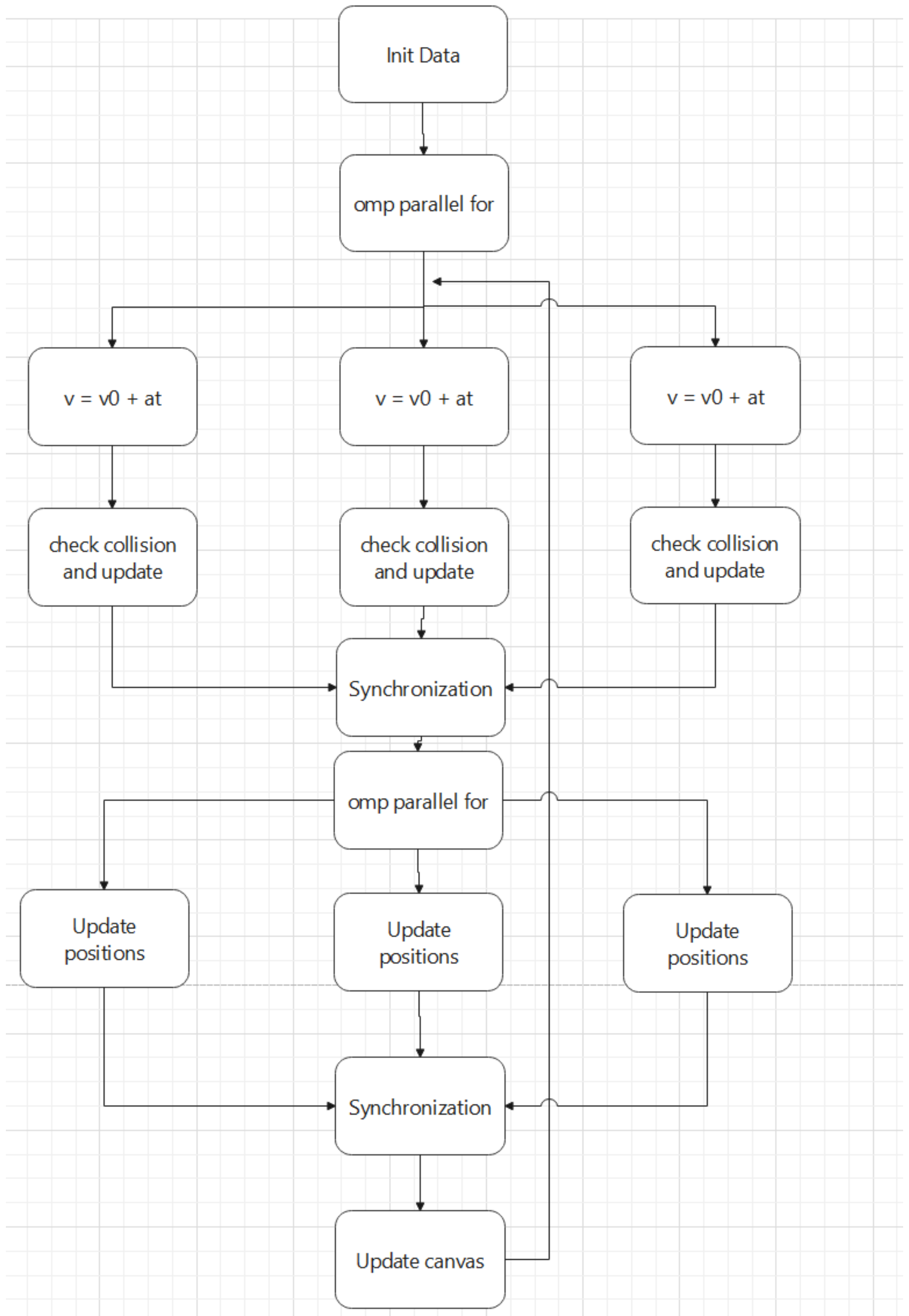

Figure 3: The Openmp Flow

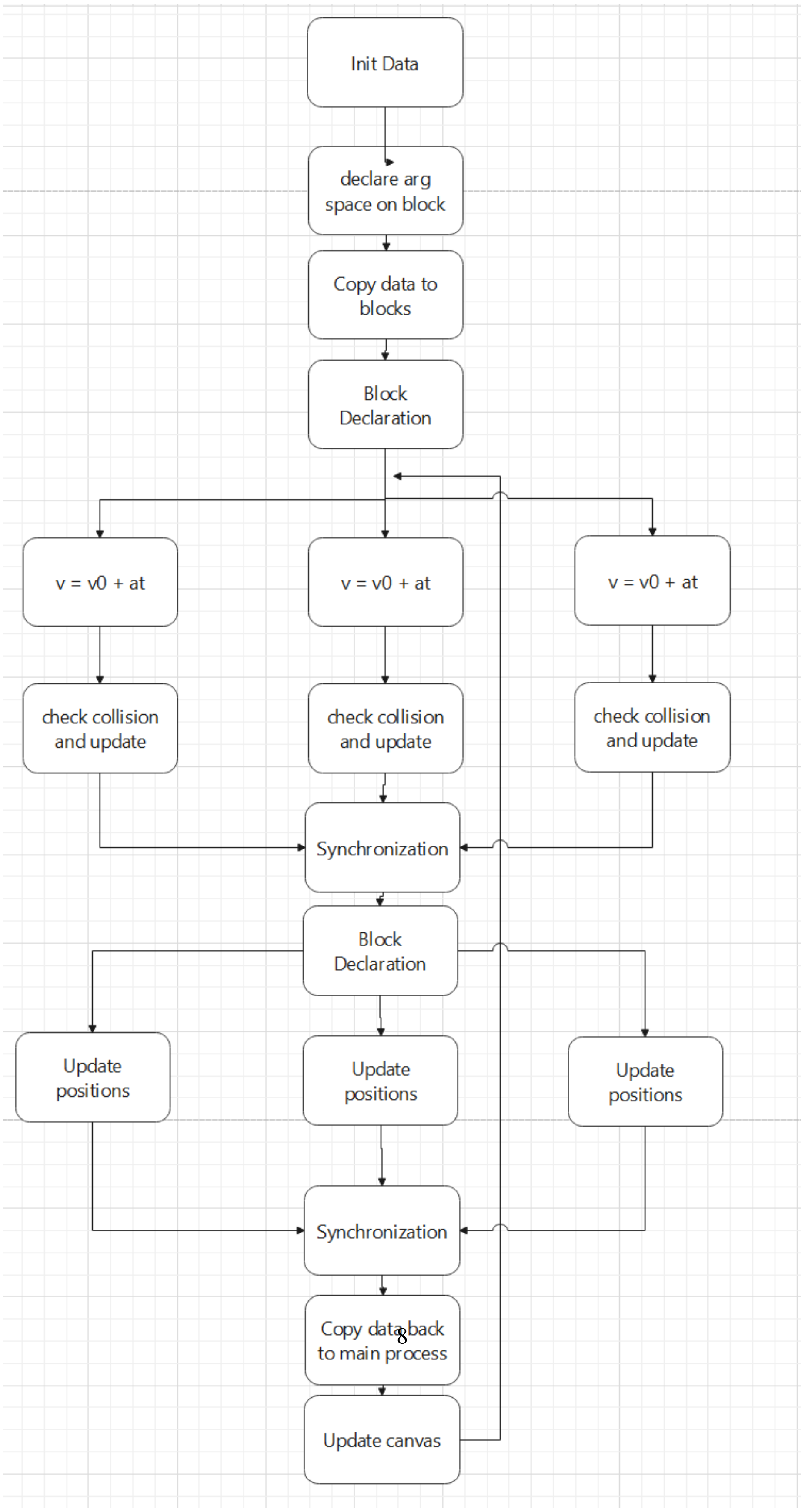

Figure 4: The CUDA-version Flow

3.2 The Implementation Details

3.2.1 Process/Thread Synchronization

Synchronization is vital to multi-process and multi-thread cooperation. In this project, synchronization can replace the lock to save time but still prevent the data racing. The MPI and the pthread libraries use different functions for synchronization. The MPI uses

```
int MPI_Barrier(MPI_Comm comm)
```

and the pthread uses

```
int pthread_join(pthread_t __th, void **__thread_return)
```

Both of the functions are to make the program wait until all processes/threads reach this point. The parallel declaration of Openmp synchronizes all processes as default.

```
#pragma omp parallel for
```

So do the parallel-computing functions of CUDA.

4 Experiment Results and Analysis

As expected, the 800*800 images are correctly plotted by sequential, MPI, pthread, Openmp and CUDA versions. The result of Openmp with 400 bodies, 100 iterations and 4 processes is shown as an example in Fig. **??**.

By varying the body number and process or thread numbers, the performance of 5 implementations are compared and analysed. The time in tables is the elapsed time of each iteration with the unit as second. The Pthread version turns out to be the optimal one. The Openmp version is slightly slower than it but really close to it, because it distributes tasks to multi-processors and the time consumed by communication increases. Compared to MPI, these two versions operate threads which are much more light-weight than processes. CUDA needs to spend extra time on the communication between the host and the GPU as well as between different blocks, which makes it slower than other versions.

The larger number of bodies consume more time, which is quite straight-forward. When the number of processes or threads is under some threshold, the time consumption of the same number of bodies reduces as the process or thread number increases. The efficiency increases as the parallelism degree increases. However, the time consumption increases again when there are too many processes or threads, because scheduling them, communications between them and some strategies like mutex locks and synchronization can take up much time.

Table 1: The MPI Performance Comparison of Different Numbers of Processes and Bodies (s/iter)

Size\Cores	1	2	4	10	20	30	40
200	0.000444	0.000332	0.000240	0.000154	0.000118	0.000098	0.001661
400	0.001681	0.000909	0.000556	0.000314	0.000215	0.000641	0.001669
800	0.006761	0.003421	0.001790	0.000844	0.000662	0.002272	0.000327
1000	0.010418	0.005300	0.002925	0.001214	0.000888	0.008356	0.000527
2000	0.041754	0.021423	0.010574	0.004531	0.003252	0.033333	0.001729

Table 2: The pthread Performance Comparison of Different Numbers of Threads and Bodies(s/iter)

Size\Cores	1	4	20	40
200	0.000626	0.000435	0.000899	0.001461
400	0.001846	0.000803	0.001121	0.001666
800	0.007052	0.002426	0.002002	0.001889
1000	0.012205	0.002987	0.002795	0.002410
2000	0.043649	0.014663	0.004964	0.006380

Table 3: The Openmp Performance Comparison of Different Numbers of Threads and Bodies(s/iter)

Size\Cores	1	4	20	40
200	0.000511	0.000750	0.003343	0.002659
400	0.001667	0.001106	0.002776	0.004086
800	0.006662	0.003242	0.002087	0.004167
1000	0.010425	0.003857	0.002547	0.004105
2000	0.041648	0.012948	0.004628	0.004396

Table 4: The Openmp Performance Comparison of Different Numbers of Threads and Bodies(s/iter)

Size	10	100	1000	10000
Time	0.003208	0.005266	0.019747	0.112

5 Conclusion

This project completes N-body Simulation with sequential, Openmp, CUDA, MPI and pthread and analysed the performance. As observed by experiments, when process or thread number is under some threshold, the increasing parallelism can improve efficiency. Among 5 implementations, Openmp and Pthread are the optimal ones, completing tasks with light-weight threads.