Senior Design
ENG EC 464

Memo

To:	Professor Pisano
From:	Vance Raiti, Austin Jamias, Sora Kakigi, Emika Hammond, Fadi Kidess
Team:	17
Date:	4/27/2024
Subject:	Individual Progress Report

---

1. Installation Details

As this is a software research project, there is no true “installation” of our products, but there is a process for setting up our projects for replication and future development. These processes are detailed, for each hardware platform we use, in this section. All source code can be found in the following GitHub repository: https://github.com/md-hpc/mdhpc

1.1 CPU Installation

Prerequisites:

- CPU with AVX2 support
- Linux OS
- bash 5.2

- GNU Make 4.4
- gcc 14.2

For each implementation (naive, cell lists, neighbor lists, SIMD cell lists, SIMD neighbor lists), simply invoke the Makefile within the corresponding directory to build the binary. It will create a binary called `sim` that can be invoked with all the command-line options described in the User Manual. The most important for replication are `--timesteps` and `--universe-size`, as these vary the problem size and allow the scalability of the programs to be tested Also, `--save` is a switch that, when enabled, makes the programs save simulation state in a CSV file called "particles.csv" so accuracy of the simulation can be assessed.

1.2 GPU Installation

Prerequisites:

- CUDA 12.5
- NVIDIA GPU
- python3
    - matplotlib
    - numpy

Generating testing PDB files

In the "src" directory, there is a python program called generate_pdb.py. Run this file with the given arguments to generate a pdb file of random particle positions: -o output file name, -n number of particles, -x number of cells in the x direction, same for -y and -z, -c cell cutoff radius

Generating scripts for running batch jobs

`script_generator.sh` generates scripts used to submit non-interactive batch jobs through `qsub` on the Boston University Shared Computing Cluster (SCC). The parameters `PROJ_DIR`, `SCRIPTS_DIR`, `TIMESTEPS`, `TIMESTEP_DURATION`, `TYPE` can be changed easily depending on what the user wants to be simulated. Additionally, when the user wants to generate new scripts, the previous ones are replaced.

Generating executable for interactive use

For each implementation (nsquared, nsquared shared, nsquared n3l, cell list, cell list n3l), simply invoke the Makefile within the corresponding implementation to build the binary. For example to generate the executable for the naive implementation use the following command: `make nsquared`. Depending on attributes of the PDB file, the user may need to modify the variables in the Makefile named “NSQUARED_FLAGS” and “CELL_LIST_FLAGS”.

Running the executable for each implementation

The executables can be run by running `batch_job.sh` or through an interactive GPU job. `batch_job.sh` is also generated by `script_generator.sh`. If you are running executables on an interactive GPU, invoke it like so: “<implementation> <input pdb file> <output csv file>”.

Generating visualizations of the simulations

To generate a gif of the simulation, run `src/viz.py <axis length> <input csv file> <output gif file>`.

1.3 FPGA Installation

FPGA requirements:

- Vitis 2023.1 for synthesis and bitstream generation
- Alveo U280 FPGA

Run the makefiles corresponding to the network layer and memory mapped interfaces (stream to memory map and memory map to stream). This will generate three IP packages that will allow the host PC to interact with the FPGA.

Run the makefile corresponding to the MD simulator to generate the simulator’s bitstream. This process usually takes under 10 hours but may take 20 hours to complete.

Copy the bitstream, host python files, and the text file containing particle initialization values to the host machine, and run the MD simulator by running the program run.py.

2. Requirements

A summary of the status of each of the engineering requirements laid out in the Team 17 PDRR. Those listed as (Final) were completely fulfilled by our product. Those listed as (Modified) were changed to better fit the needs of our customer and are explained below. Those listed as (Eliminated) were removed for practicality and are explained below:

1. (Modified) Design must accept any cubic periodic simulation space as input
2. (Final) Design must correctly compute the dynamics of particles in simulation
3. (Final) Design should make use of the scalable ring network architecture as introduced in Wu 2024.
4. (Modified) Design must map to computers of arbitrary number and size
5. (Modified) Design should maintain strong scaling: MD simulation speed is proportional to the total compute power available
6. (Final) Design must include implementation for FPGA
7. (Final) Design must include implementation for GPU
8. (Final) Design must include implementation for CPU
9. (Eliminated) Design must include implementation for ASIC

Explanation for modified or eliminated requirements are:

1. Design must accept any cubic periodic simulation space as input

Due to memory constraints on some of our accelerators (GPU, FPGA), we are only able to simulate a limited number of particles. Each accelerator has the following constraints:

	CPU	GPU	FPGA
Maximum # of particles	1e8	1.3e5	2e3

Table 1: Maximum Particles per Simulation for Hardware Platforms

4. Design must map to computers of arbitrary number and size

Our GPU and CPU implementations will scale to arbitrary computer size. However, due to practical constraints, we did not implement multi-node MD for any of our platforms, and, due to the highly hardware-dependent nature of FPGA code, did not produce an FPGA implementation that scales arbitrarily with processor size.

5. Design should maintain strong scaling: MD simulation speed is proportional to size of computers used

Our GPU and CPU implementations possess strong scaling with respect to computer size, but, since they have no scaling with processor count, do not have strong scaling with respect to processor count. Likewise, FPGA does not have strong scaling because it does not scale.

9. Design must include implementation for ASIC

Due to the momentous amount of effort that would be required to design an entirely new hardware platform specifically for MD, and given the lack of prior experience in the area from our team, we decided early in the project that this feature was out of scope for this project.

3. Evaluation

For each implementation on each hardware platform, we benchmarked performance on a specific hardware node (CPU: SCC AVX2 16 core, GPU: Nvidia A40, FPGA: idk). The performance of each is given below:

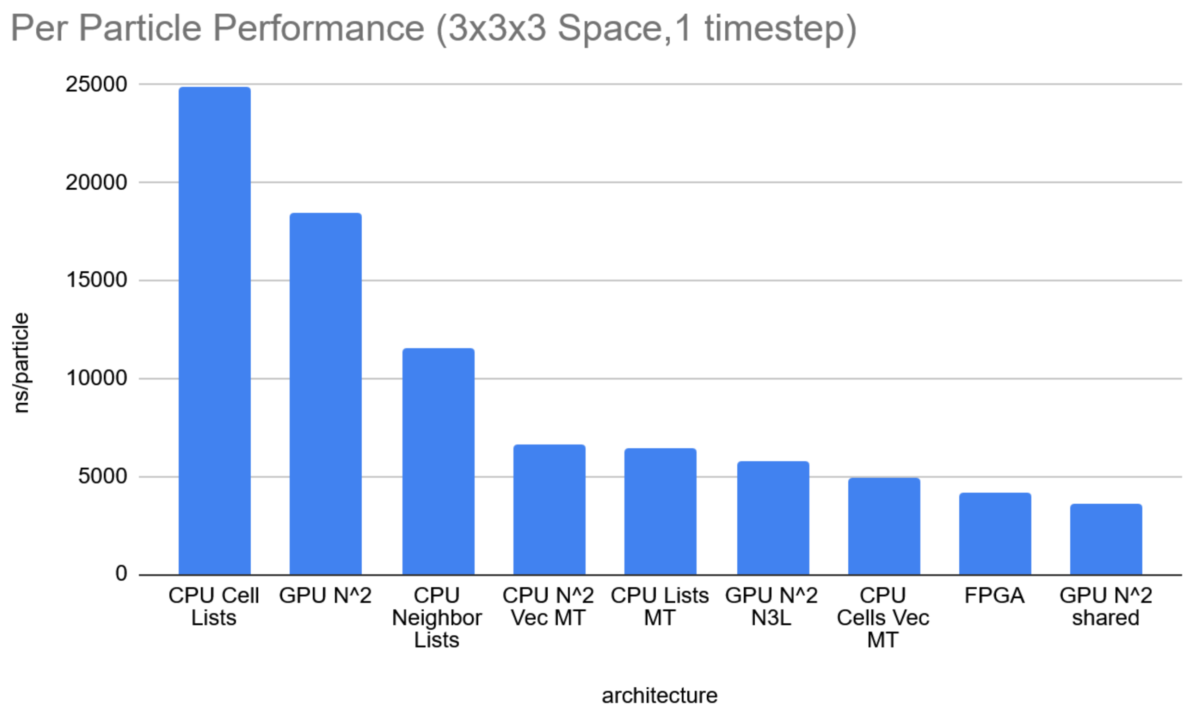

Figure 1: Performance of various molecular dynamics simulations

The best attainable performance per hardware platform, along with other statistics, are given below:

Requirement	CPU	GPU	FPGA
Maximum error per particle (from serial baseline)	< 1e-5	< 1e-4	< 1e-5
Time per particle per timestep	500 ns	100 ns	150 ns
Maximum particles per simulation	1e+8	128K	2k

Table 2: Statistics of best implementation on each hardware platform

Finally, for each of the engineering objectives, we state how it was fulfilled:

1. Design must accept any cubic periodic simulation space with as many particles as can be practically fit on-chip

Each program for each platform has a command-line option that allows arbitrary PDB (protein data bank) files to be passed in as the simulation initial conditions as long as the particles in-memory are not so numerous that they cannot be represented with the finite memory of the accelerator.

2. Design must correctly compute the dynamics of particles in simulation

Each program for each platform has the ability to log simulation state at each timestep, allowing us to compare with a baseline to determine correctness. The data in Table 2 shows the maximum observed error, which is at or below our tolerable error of 1e-4.

3. Design should make use of the scalable ring network architecture as introduced in Wu 2024.

While GPU and CPU have largely fixed hardware configuration and so cannot make use of the ring node technology, our FPGA implementation uses ring nodes to achieve high throughput interconnects.

4. Design must map to computers of arbitrary size

The CPU is able to dynamically determine how many hardware threads are available and scale appropriately. The GPU application will scale due to bookkeeping by the CUDA runtime.

5. Design must include implementation for FPGA

Our FPGA implementation is still a WIP (the figures given in figure 1 and table 2 are estimates).

6. Design must include implementation for GPU

We have a working GPU implementation that we have tested extensively on the Nvidia A40 on the SCC

7. Design must include implementation for CPU

We have a working CPU implementation that we have tested extensively on the 16-core AVX2 CPUs on the SCC.