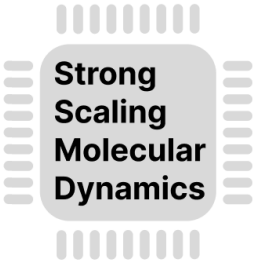
Boston University
Electrical & Computer Engineering
EC463 Capstone Senior Design Project

First Prototype Test Report

Strong Scaling Molecular Dynamics



by

Team 17

Vance Raiti, Austin Jamias, Emika Hammond, Sora Kakigi, Fadi Kidess

Vance Raiti vraiti@bu.edu
Austin Jamias ajamias@bu.edu
Emika Hammond eth@bu.edu
Sora Kakigi skakigi@bu.edu
Fadi Kidess fadik@bu.edu

Testing Environment

Python Prerequisites

- Numpy 1.26
- Matplotlib

Emulator code (https://github.com/md-hpc/hls)

Hardware emulator

- emulator.py
- phase{1,2,3}.py

Software-native implementation

- verify.py

Visualization script:

- viz.py

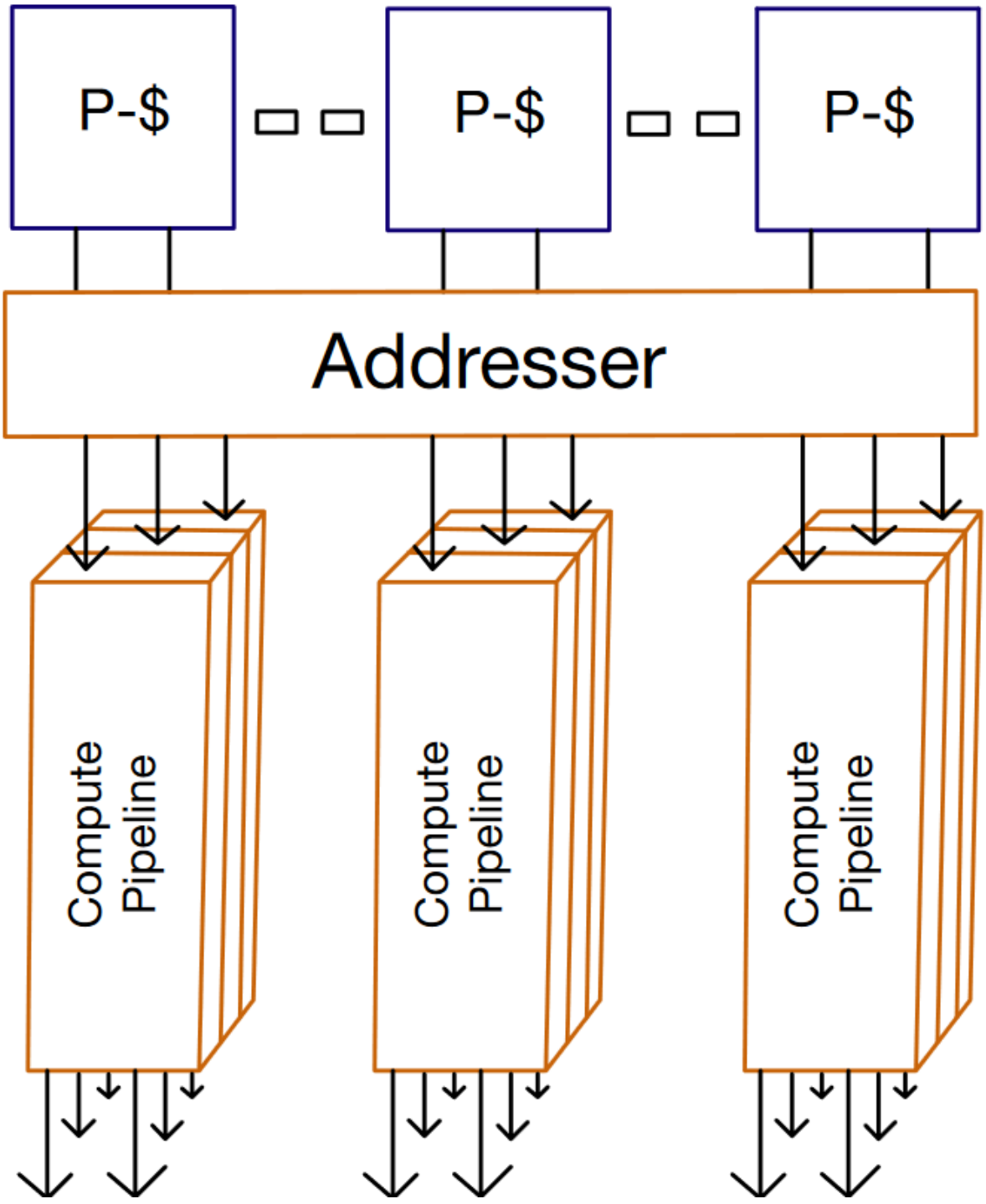


Fig 1: Force Computation Architecture

Set-Up

The prototype is a Python script that models the RTL computation found on an FPGA using a set of classes that model FPGA combinational logic and register state update using Python functions. Special classes are used for memory (BRAM) and module-external registers.

The emulator is divided into three “phases:”

1. **force computation** - filter and compute particle force interactions
2. **velocity update** - combine fragments into particle velocities
3. **motion update** - calculate the new particle positions and update for next timestep

Each phase gets a dedicated source file that defines its logic components and connections to memory (e.g., force computation will need access to the position and acceleration caches). An external “control unit” (defined in the top-level emulator.py) enables each of these phases sequentially until a time step is complete.

The simulations are configured by a set of constants found in common.py. These parameters can be divided into three categories: **(1) hardware, (2) simulation, and (3) physics**. **(1) hardware parameters** vary the scale of the hardware being emulated (how many compute pipelines are dedicated to which cells, etc). **(2) Simulation parameters** vary what is being simulated (how large is the simulation universe, the initial state of the particles, etc.). Finally, **(3) physics parameters** vary the physical properties of the particles: epsilon and sigma for the Lendard-Jones force computations.

To ensure the correctness of the emulator, a verification routine found in verify.py computes the next timestep’s positions using a software-native algorithm that is a direct implementation of the high-level computation performed by all MD simulators. It will compare the positions computed during the previous timestep by the emulator and by the verifier to determine if any errors were made in the computation. This software-native algorithm is easy to reason about and a standard in MD, so we (the team and our customer, Martin Herbordt) accept it as ground truth.

Testing Procedure

Three sets of experiments are run to demonstrate the desired characteristics of this emulator: **correctness, generalizability, and scalability**.

1. To demonstrate **correctness**: a small (300 particle and 27 cells) simulation will be run for 20 timesteps. At each timestep, the verification routine will confirm the correctness of the emulator.
2. To demonstrate **generalizability**: a range of simulation sizes (300, 400, 500 particles and 27, 64 and 125 cells) will run for 2 timesteps, confirming that the emulator can simulate arbitrary systems.
3. To demonstrate **scalability**: a range of hardware configurations (1 to 6 simultaneous reference particles and 1 to 6 simultaneous reference cells) will be evaluated for a fixed simulation (300, 2000 particles and 27 cells) that will run for 1 timestep.

A plot showing the number of emulated cycles required for each hardware configuration will demonstrate that increased hardware improves performance on a fixed problem size, demonstrating scalability.

The shell scripts to run the experiments for correctness, generalizability, and scalability are contained in the Git repository as run-correctness, run-generalizability, and run-scalability respectively. As long as the testing environment has the Python prerequisites mentioned above installed, the experiments can be run anywhere (although they might take a while).

Measurable Criteria

1. The percent error between simulated position update computed by the emulator and verifier should never exceed $1x10^{-7}$ (attributable to floating-point roundoff error)
2. The emulator should successfully simulate at least 20 timesteps of a simulation without exceeding error in criterion (1)
3. The emulator should successfully simulate at least 9 different simulations of varying size and particle count without exceeding error in criterion (1)
4. The emulator should demonstrate approximately linear scaling of emulated performance with increased hardware allocation on a fixed problem size

Results and Discussion

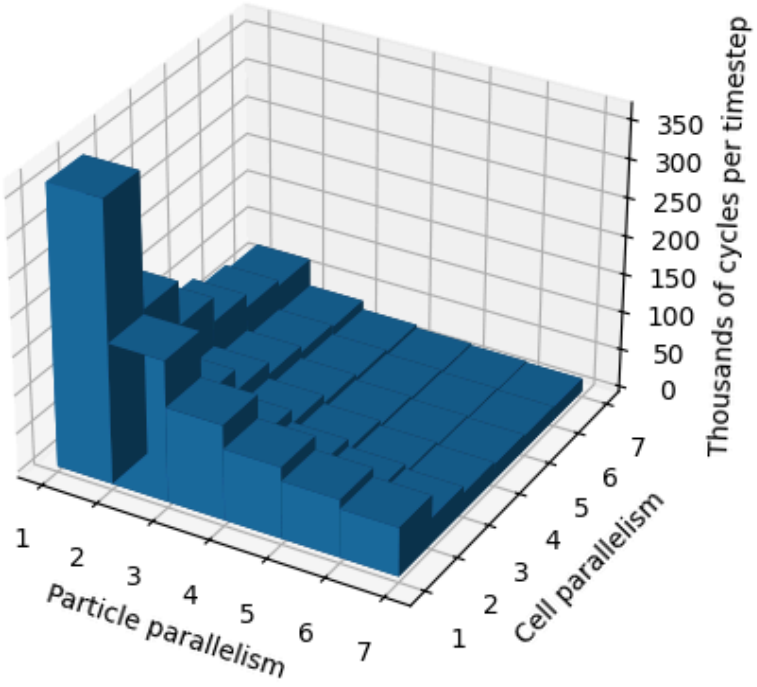


Figure 1: Scalability of MD Engine

The emulator achieved our correctness, generalizability, and scalability requirements. Correctness and generalizability scripts were both executed while maintaining no more than $1x10^{-7}$ percent error between emulator and verifier computations. Maximum error averaged around $1.6x10^{-11}$ percent from the target computations. Perfect scalability would indicate that the number of cycles was inversely proportional to the number of compute pipelines. We found the correlation coefficient of this relationship to be 0.997 for the 300-particle simulation and 0.999 for the 2000-particle simulation.

The architecture we used for this emulator should easily map to FPGA technology, as the only assumptions we made about the hardware were that functional Python routines can model combinational FPGA logic and that BRAMs have dual read/write IO ports.

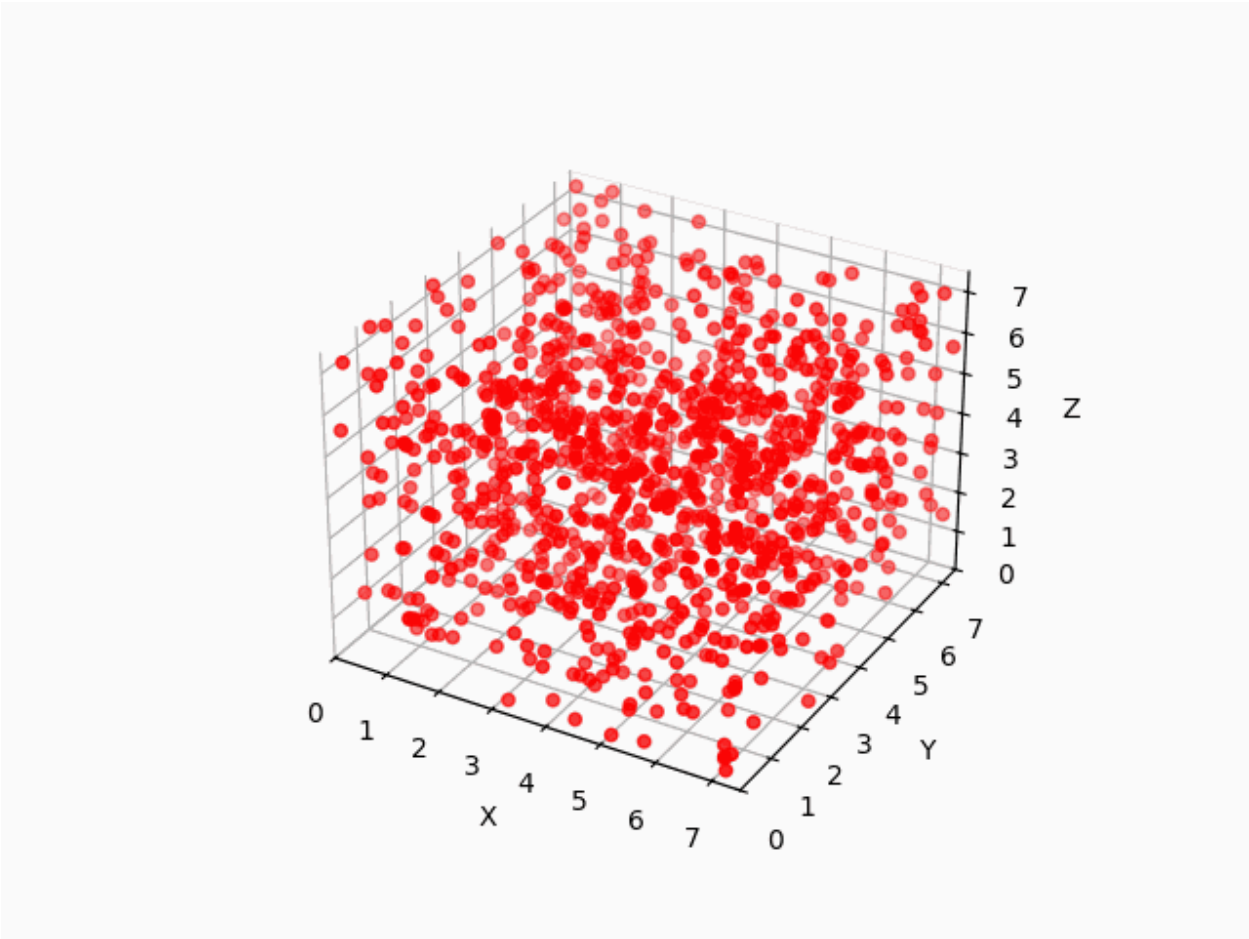


Fig 2: Sample MD Simulation Visualization of 300 Particle Simulation