

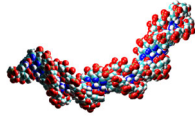


Molecular Dynamics Simulation on the Merrimac Streaming Supercomputer

Ankit Garg, Yanan Zhao, Mattan Erez, Abhishek Das, Eric Darve



Background



DNA molecule

Streaming supercomputers will have wide applications in complex scientific computing. Our work with molecular dynamics simulation is a preliminary step in using streaming supercomputing to model the **folding of human proteins**. Learning more about this process will have a tremendous impact on biology and medicine and will help medical researchers understand diseases and find cures.

Numerical ODE in Molecular Dynamics

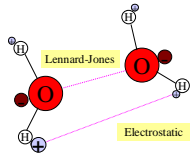
The approach we have decided to take is Molecular Dynamics (MD).

Newton's equations of motion

$$m_i \frac{d^2 \vec{r}_i}{dt^2} = -\nabla_{\vec{r}_i} U(\vec{r}_1, \dots, \vec{r}_N) = \vec{F}(\vec{r}_i)$$

Potential energy U is defined as the sum of two terms:

- Lennard-Jones potential, which decays as $\frac{1}{|\vec{r}_i - \vec{r}_j|^6}$ Used for oxygen atoms only.
- Electrostatic potential, which decays as $\frac{q_i q_j}{4\pi\epsilon_0 |\vec{r}_i - \vec{r}_j|}$ For all charged particles.



Potential Energy for Water Molecules

The Ordinary Differential Equations (ODE) can be solved with the **Leap-frog scheme**.

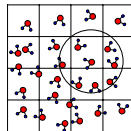
$$\vec{r}_i(t + \Delta t) = \vec{r}_i(t) + \Delta t \vec{v}_i(t) + \frac{\Delta t^2}{2} \vec{F}(\vec{r}_i(t))$$

$$\vec{v}_i(t + \frac{\Delta t}{2}) = \vec{v}_i(t - \frac{\Delta t}{2}) + \Delta t \frac{\vec{F}(\vec{r}_i(t))}{m_i}$$

Using this method, it is possible to simulate the complex trajectories of atoms and molecules for very long periods of time. However, only the advent of more powerful supercomputers will allow studying molecules over time scales which are biologically and experimentally relevant.

In order to start with a simple test case, we simulated a **box of water molecules**.

Neighbor List Technique



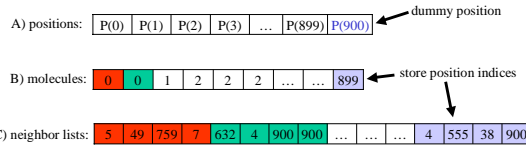
Applying cutoff

The most expensive part of the simulation is the computation of the forces $\vec{F}(\vec{r}_i)$.

A **cutoff** is applied so that all particles which are at a distance greater than a cutoff radius do not interact. Particles within the cutoff radius are part of the neighbor list.

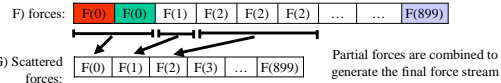
One Timestep of the Force Calculation

1. Load initial input data into streams.



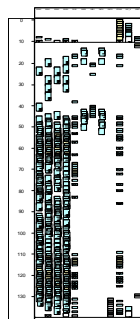
Each neighbor list has the same length. Molecules with extra neighbors have multiple neighbor lists. Molecules with too few neighbors are given dummy neighbors.

- Gather positions into new streams (D and E) indexed by B and C.
- Compute the forces between each molecule and the molecules in its neighbor list(s). Store the forces in a new stream F.
- Scatter the forces into a stream indexed according to the original position stream (A).



Optimization: The Arithmetically Intensive Kernel

The kernel performs all the arithmetic operations of the force calculation. The schedules below track the operations within one iteration of the kernel. The vertical axis marks time in number of cycles. The horizontal axis marks each functional unit in the SSS architecture: columns 1-4 are for addition and multiplication, column 5 is for division and square root. Each rectangle represents an operation underway.

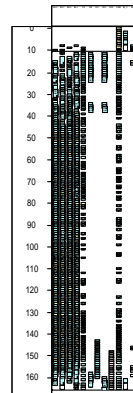


Left-hand side: No optimizations

Right-hand side: Software pipelined and loop unrolled.

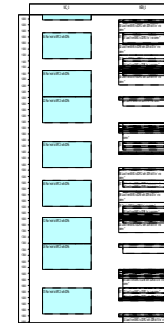
- Software pipelining: Computes resource-independent parts of several kernel iterations within one iteration.
- Loop unrolling: Reduces two complete kernel iterations into one, performing most operations from both in parallel. Consequently, the schedule to the right is equivalent to two schedules from the left placed one after the other.

Result: The same kernel is 1.7 times faster after optimization.



Optimization: The Full Application

The two schedules below track the main processes in the overall application. The vertical axis marks time in number of cycles. The left column tracks chunks of the kernel currently in progress. The right column tracks I/O stream operations.

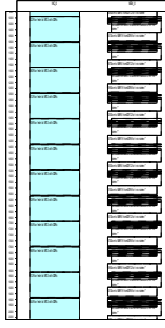


Left-hand side: Automatically double-buffered.

Right-hand side: Manually striped, MARs increased.

- Double-buffering and Striping: Our application has only one kernel, but it requires more data from input streams than can be loaded into the SRF at one time. Hence either the computer (the left case) or the programmer (the right case) splits the input streams into strips that are fed to the SRF piecemeal, causing the kernel to run piecemeal as well.
- Memory Address Registers (MARs): channels between the memory and the SRF through which streams flow.

Result: Manually optimized code is 1.35 times faster than automatically optimized code.



Merrimac vs. Pentium

The execution is hindered by the memory operations, and does not fully utilize the functional units. This can be seen by looking at the number of cycles separating calls to the kernel. This is due to the fact that the computation cannot start until all the molecules have been gathered and filtered.

This problem will be solved with SSS, whose **bandwidth memory system is larger than Imagine** by a factor of about 2. We also introduced a **stream cache** on the SSS for an additional bandwidth amplification. These two elements will eliminate the stalls.

Future Work

- Changes will be made to the algorithm used by the profiler so that better performance is achieved with automatic optimizations of the code.
- The full GROMACS code with several time-steps of force calculations will be run on Merrimac and the performance tested.
- Actual computations for **complex proteins** will be run on the NV30 and the SSS simulator.

