Performance Analyses of a Parallel Verlet Neighbor List Algorithm for GPU-Optimized MD Simulations

Andrew J. Proctor¹, Tyson J. Lipscomb¹, Anqi Zou¹, Joshua A. Anderson³, and Samuel S. Cho¹,²
Wake Forest University, Departments of Computer Science¹ and Physics²
University of Michigan, Department of Chemical Engineering³

Definition and Background
- Two major classes of interactions to calculate:
  - Bonded: bonds, angles, dihedrals
  - Pairwise: van der Waals, electrostatics
- O(N²) calculation
- Ideal for parallel computation, e.g., on a GPU architecture

Algorithm is Highly Parallelizable
- Each bead’s analysis, velocity, and force are independently calculated at each timestep.
- Ideal for parallel computation, e.g., on an architecture

Solution: Assign each interaction to its own individual thread.

GPU-Optimized Parallel Neighbor List Algorithm

Step 1: Perform key-value sort on GPU using CUDPP library.
  - Member List as keys and Master List as values.
  - Groups members of Neighbor List together with others.
  - Keys are binary flags, so a 1-bit sort suffices.

Step 2: Perform parallel scan using CUDPP.
  - Counts the total number of TRUE values in Member List, determining how many entries are in Neighbor List.

Step 3: Update Neighbor List to point to the first num_NL values of Master List.

Original Neighbor List Algorithm is Not Readily Parallelizable
- Each iteration is dependent upon the results of previous iterations.
- Threads would be dependent upon one another.
- Cannot parallelize!

Problems for Parallelization
- Each iteration is dependent upon the results of previous iterations.
- Threads would be dependent upon one another.
- Cannot parallelize!

Performance Comparisons to GPU-Optimized HOOMD MD Simulation Suite

- Comparison of execution times of the SOP model implemented in our MD simulation code as compared to the SOP model implemented in HOOMD for system sizes up to ~1,000 beads, which was the maximum hardware limitation of the NVIDIA Tesla C2070.
- The same comparison for systems up to ~10,000 beads, which was the memory footprint limit of our simulation code on the NVIDIA Tesla C2070.

Conclusions
- Molecular dynamics simulations can be highly optimized using NVIDIA’s CUDA API along with the CUDPP and CURAND GPGPU libraries.
- Though memory transfers can cause severe bottlenecks, compression of data can significantly reduce overhead.
- Even non-parallel algorithms can be optimized to a high degree by developing new parallel approaches.
- There exists an N-dependent GPU vs. CPU performance speed-up (or –down).
- Direct comparisons of the SOP model implemented in our GPU-optimized software as compared to HOOMD demonstrates that our code can accommodate biologically relevant system sizes.

Acknowledgements
- Wake Forest University Computer Science Graduate Fellowship for Excellence (TJL)
- National Science Foundation
- NVIDIA Academic Partnership (SSC)
- Wake Forest University Research Fund (SSC)