

Brown Deer --- **Technology**

White Paper

Porting the LAMMPS/EAM benchmark to OpenCL™ with limited code modifications

David A. Richie

March 29, 2011

Introduction. Demonstrating the use of GPUs for technical computing normally focuses on the use of *GPU-tuned algorithms* for accelerating applications over the baseline performance on a CPU. Although speedups of as much as 100x or greater are sometimes reported, more realistic speedups are in the range of 2x to 10x when careful consideration and effort is given to comparable optimizations for the CPU. The acceleration of GPU-optimized code often neglects the realities of existing production HPC codes in terms of acceptable modifications, where absolute performance is only one of many factors considered in software design decisions. This white paper discusses the initial results from an investigation of a pre-defined benchmark for a production molecular dynamics code (LAMMPS) using OpenCL™ with limited modifications to the original source code, leaving the algorithm unchanged.

Molecular Dynamics. One of the most important methods used for chemistry and material simulation is molecular dynamics (MD) in which the motion of atoms and molecules are simulated as a collection of particles subject to inter-particle forces. Molecular dynamics has been the target of many GPU investigations. As an example, in previous work (2008) we accelerated the LAMMPS rhodopsin benchmark with a FireStream 9170 using Brook+. This effort required significant code modifications including a complete reformulation of the underlying pair-potential algorithm to match the programming model and the GPU.

LAMMPS¹ is a very popular molecular dynamics code used in production HPC environments by many researchers. This popularity is most likely attributable to its stability and functionality, good scaling on large systems, and *accessible source code* that allows the researcher to read, understand and modify the physics in the simulations. Absolute per-core performance at the expense of these attributes may be interesting, but is not necessarily desirable in the evolution of the code. In the work described here, sufficient modifications are made to the LAMMPS code to allow the EAM (embedded atom model) benchmark to be executed on OpenCL devices (GPUs and CPUs) within the constraints that these modifications should be limited, portable, and not obscure the physics by altering the existing algorithm.

¹ <http://lammps.sandia.gov>

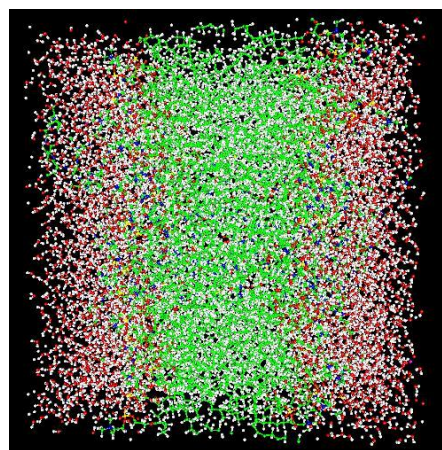


Figure 1. 32,000-atom LAMMPS molecular dynamics simulation of a rhodopsin protein in a solvated lipid bilayer with the particle-particle interaction calculated on an AMD FireStream 9170. Obtaining these results in 2008 required significant modifications to the source code in order to address the programming model and GPU.

Practical Challenge. Since many efforts such as this focus on the GPU implementation, it is important to stress that in this investigation there is no “GPU implementation”. As an increasing number of hybrid CPU/GPU computing platforms become available, many application developers and users may not be interested in lengthy efforts to port their code to a new architecture, but instead will search for the shortest path that allows them to take practical advantage of these systems. This investigation concerns the “shortest path” using OpenCL.

The specific constraints imposed here on source code modifications include requiring: (i) no change in the algorithm in terms of memory access pattern and computation, (ii) no GPU-specific optimizations, (iii) no single-precision “tricks” or other approximations, regardless of how valid they may be, and (iv) no device-specific code variants, i.e., the “physics” should be implemented once in the source code. With this approach, the full double-precision algorithm, as implemented in the original LAMMPS code, is used for the OpenCL kernels that implement the computation.

OpenCL and STDCL. OpenCL is an industry standard² programming API for parallel programming of heterogeneous computing platforms. Although OpenCL is commonly viewed as a GPU programming

² <http://www.khronos.org/opencl>

Porting the LAMMPS/EAM benchmark to OpenCL with limited code modifications

language, its applicability is more general and specifically includes multi-core CPUs. OpenCL is well-suited for the task of providing a device and vendor neutral implementation since support is available for all relevant CPU and GPU processors. One deficiency with OpenCL in the context of this investigation, and for HPC applications in general, is that the API is designed with such generality to suit a wider industry, that the practical use of OpenCL can be complex and tedious – it is a verbose API designed to control a great many things that typical HPC users will have no interest in worrying about. This has served as the motivation for the STDCL interface.

STDCL³ (STandarD Compute Layer) is a simplified programming interface for OpenCL designed to support the most typical use-cases in a style inspired by familiar and traditional UNIX APIs for C programming. The interface is provided as part of the COPRTHR (CO-PRocessing THReads) SDK⁴ developed by Brown Deer Technology and freely available under an open-source license (LGPLv3). The STDCL interface provides more than wrappers to direct OpenCL calls, and includes support for default OpenCL contexts, conventional memory allocation of device-sharable memory, and OpenCL event management. The use of STDCL is critical for achieving the objective of introducing minimal source code modifications in porting LAMMPS to OpenCL.

Modifying LAMMPS. Modifications were applied to the LAMMPS code cloned from the project's git repository on 22 March, 2011. Modifications were made so as to port the EAM pair potential and nearest neighbor calculation to OpenCL, which accounted for more than 98% of the computation for the unmodified code. Focusing on only these two components of the calculation requires communicating data across the PCIe bus at each step when using a GPU, and thus presents an issue worth exploring in future work.

It should be noted that LAMMPS presently provides several pair potentials that have been ported to Nvidia GPUs using CUDA. The modifications discussed here have not used the existing CUDA code in any way. All modifications begin with the original CPU source code provided in the current release.

A summary of the source code modifications⁵ is as follows:

1. OpenCL Initialization. The use of the STDCL interface eliminates almost all requirements for initialization, requiring only that the `stdcl.h` header be included in the appropriate files and that the application is linked to the `libstdcl.so` library. The only explicit initialization required is that the OpenCL kernels for the nearest neighbor calculation and the EAM pair potential calculation must be loaded prior to use, which is accomplished with the STDCL `clopen/clsym` calls amounting to a few lines of code.
2. Memory Allocation. The LAMMPS memory allocation calls `smalloc`, `smrealloc` and `sfree` are modified to use the STDCL equivalents of `malloc`, `realloc` and `free`. By default, memory allocation is detached from any OpenCL context, so that this modification has no impact on the code for non-OpenCL routines. Specific data that is to be shared between the host platform and an OpenCL device must be attached to an OpenCL context, and this is accomplished with a single STDCL `clmattach` call. The overall result is that the modifications required to utilize device-sharable memory are nearly transparent.
3. EAM Pair Potential Calculation. This represents the most significant modifications. The existing EAM pair potential code was copied to create the variant "eam/ocl" and integrated into the simulation package as a valid option in the input files. The actual compute function was removed and replaced with host code to control memory synchronization and computation on the OpenCL device. The algorithm was partitioned into three kernels corresponding to the three consecutive main loops over atoms in the original code, with the algorithm left unchanged.
4. Nearest Neighbor Calculation. A variant of the nearest neighbor algorithm "full_bin" was created so as to execute on the OpenCL device. A copy of the original algorithm was moved into an OpenCL kernel, and the routine "full_bin_ocl" was created to control memory synchronization and the computations on the OpenCL device.

³ http://www.browndeertechnology.com/coprthr_stdcl.htm

⁴ <http://www.browndeertechnology.com/coprthr.htm>

⁵ The modified source code is available at <http://github.com/browndeer/lammps-ocl>

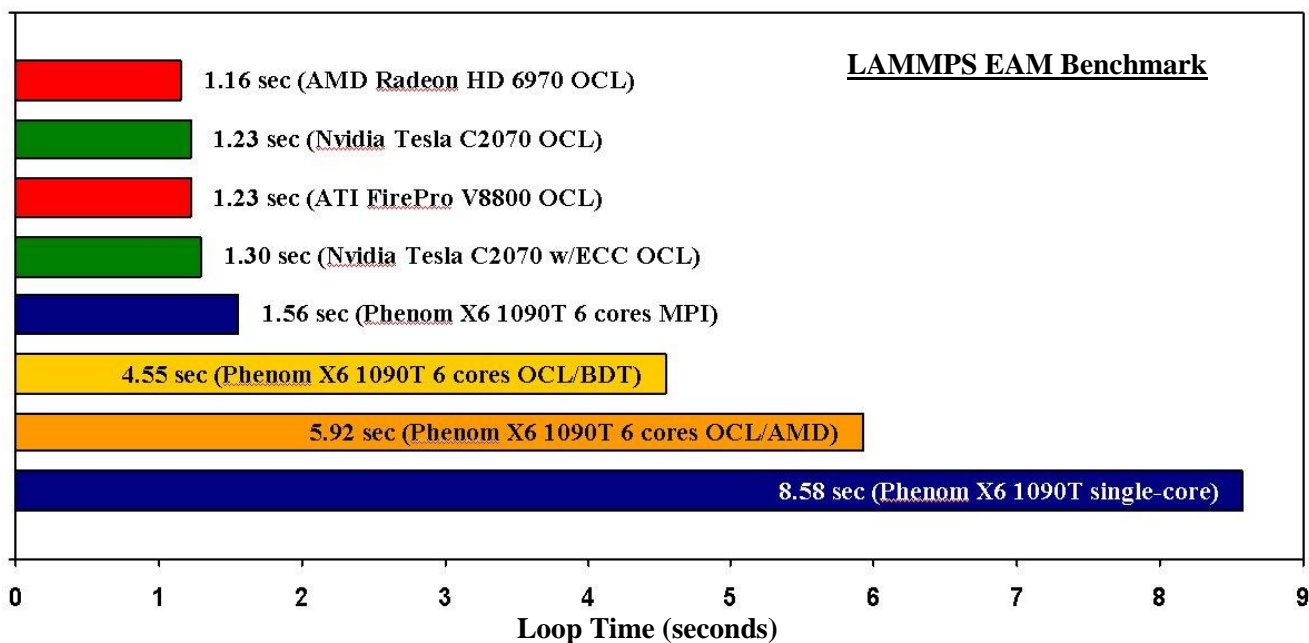


Figure 2. Total loop time reported for the LAMMPS EAM benchmark which models 32,000 copper atoms for 100 time-steps using the embedded atom model (EAM) for inter-atomic forces. GPU benchmarks use the same OpenCL implementation using limited source code modifications to the original LAMMPS CPU implementation with no change in the original algorithm. The same OpenCL implementation is used to target the 6-core CPU as an OpenCL device using the AMD and BDT OpenCL run-time support for x86_64. As a reference single-core and 6-core MPI benchmarks using the unmodified LAMMPS code are also provided.

An important distinction between the original CPU code and the OpenCL implementation is that the original EAM pair potential uses a half neighbor list, which provides an optimization by reducing the computation by approximately a factor of 2. For the OpenCL kernels, a full neighbor list must be used to avoid write-back data collisions. This is not considered a modification to the CPU algorithm since the use of a full neighbor list is a valid option in LAMMPS, and the use of the full list puts the OpenCL implementation at a disadvantage. Simply put, the OpenCL benchmarks perform twice the work as compared to the original CPU code. This is not factored into the comparison of benchmarks, since from a users perspective, what matters is the ultimate performance, not whether one implementation performs more (redundant) computations to achieve the same results.

Benchmarks. Figure 2 shows the results for the LAMMPS EAM benchmark for various hardware and software configurations. All benchmarks were run on two systems with very similar hardware

configurations.^{6,7} Each system used an AMD Phenom II X6 1090T 6-core processor. Although this is a “desktop part” it represents a very high-end 6-core processor, and it is worth noting that the CPU benchmarks reported here are faster than any benchmarks reported on the LAMMPS website up to 8 cores. Nevertheless, the results should not be interpreted as an ultimate CPU-GPU “shoot out” since this would require a comparison with the latest 8- and 12-core server chips. However, the CPU benchmarks very likely represent real-world systems in terms of performance at this time.

The results for all OpenCL benchmarks, as reported by LAMMPS in terms of aggregate metrics such as energies and pressures, were identical to those reported

⁶ System 1 configuration: ASUS M4A79T, AMD Phenom II X6 1090T BE @ 3.2 GHz, 16GB G.SKILL DDR3-1600 (9-9-9-24 timing), Linux CentOS 5.4 x86_64, AMD/ATI FirePro V8800, AMD Radeon HD 6970, AMD Catalyst driver version 11.1, ATI Stream SDK v2.3, COPRTHR SDK v1.2 (pre-release)

⁷ System 2 configuration: ASUS M4A89TD Pro, AMD Phenom II X6 1090T BE @ 3.2 GHz, 12GB G.SKILL DDR3-1600 (8-8-8-24 timing), Linux CentOS 5.5 x86_64, Nvidia Tesla C2070, Nvidia driver version 270.27, CUDA Toolkit 4.0.11-RC, COPRTHR SDK v1.2 (pre-release)

Porting the LAMMPS/EAM benchmark to OpenCL with limited code modifications

for the unmodified code, consistent with the use of the identical double-precision algorithm. The same code is used for each OpenCL device (GPU and CPU) without modification. The only device-specific tuning was the choice of workgroup size, which is a run-time parameter that is expected to impact performance. Reported times reflect the fastest times observed for the tested values.

The benchmarks for all GPUs exhibit remarkably good performance considering the constraints imposed on the source code modifications and the use of the algorithm designed for a serial CPU core. This algorithm places a heavy burden on the memory architecture, being dominated by out-of-order memory access patterns. Without tuning the algorithm for the GPU architectures, each GPU outperformed a high-end 6-core CPU (using all cores) by a decisive margin.

The fastest loop time was observed for an AMD Radeon HD 6970 (Cayman XT), followed by the Tesla C2070 (Fermi) and the AMD/ATI FirePro V8800 (Cypress) posting nearly identical times that were only marginally slower. The Tesla C2070 is unique in providing support for ECC memory, which is critical in many HPC environments. With ECC memory enabled, the C2070 exhibits only a slight performance penalty, and still decisively outperforms the CPU.

The same code used for the GPU benchmarks can be executed on the CPU as an OpenCL device without modification. Benchmarks were run using the AMD OpenCL run-time (OCL/AMD) as well as the open-source OpenCL run-time provided by the COPRTHR SDK (OCL/BDT). For this benchmark, the BDT run-time⁸ proved to be faster than the vendor implementation by a significant margin. Nevertheless, performance was not close to that of the original LAMMPS code using MPI on 6 cores. This result is not surprising considering the significant effort has been invested in the scaling of the LAMMPS code using MPI. In light of the fact that the OpenCL implementation is performing twice the work due to the use of a full neighbor list, the comparison looks more favorable, however MPI still wins over OpenCL on the CPU. These results could change with improvements to the OpenCL run-time or the use of

atomic operations to permit the use of a half neighbor list.

Conclusions. The ability to run the same code across GPUs and CPUs from different vendors reflects a significant advance in multi-core and many-core technology. With limited source code modifications, the LAMMPS EAM benchmark was ported to OpenCL to allow for cross-device and cross-vendor benchmarks using one implementation that retains the original algorithm targeting the CPU. Without any GPU optimizations, high-end GPUs from AMD and Nvidia are able to outperform a high-end 6-core CPU using the existing MPI parallelization in LAMMPS. At minimum, these GPUs should be able to match if not outperform the highest performance CPUs found in real-world systems. For OpenCL targeting CPUs, further work is needed to match the performance of MPI. The limited source code modifications used in the port to OpenCL provide a baseline for ongoing work exploring the use of multiple devices and device-specific optimizations.

⁸ The BDT OpenCL implementation relies on the AMD front-end compiler (clc) to produce LLVM code, and the implementation is not yet fully conformant to the entire OpenCL 1.0 specification. Nevertheless, it is sufficient for treating a non-trivial OpenCL application and exhibiting good performance.