Software Design for Many-Core Processors: Molecular Dynamics Simulations

David Richie
Brown Deer Technology

October 26th, 2010
Outline

• Inspiration:
  • GPU-acceleration of LAMMPS (2008)
  • GPU-acceleration of two-electron integrals for quantum chemistry (2009)

• Code design project (“Auburn Lake”)
  • Software Design for GPU accelerators many-core processors
  • Step 1: Choose a terrible algorithm for “GPU acceleration”
    • Molecular dynamics with a short-range three-body potential
  • Re-writing our codes – why, when, where and how
  • Rethinking the algorithm
  • Benchmarks – Cypress beats Thuban @ 3.2 GHz, game over?
  • A path forward: OpenCL – write once, run everywhere
Molecular Dynamics: LAMMPS

- Fundamental technique for molecular modeling
- Simulate motion of particles subject to inter-particle forces
- LAMMPS is open-source MD code from DOE/Sandia
- Goal: accelerate inter-particle force calculation

- Rhodopsin Protein Benchmark (most difficult)
- Details: All-atom rhodopsin protein in solvated lipid bilayer with CHARMM force field, long-range Coulomb via PPPM, SHAKE constraints, system contains counter-ions and a reduced amount of water
- Benchmark: 32,000 atoms for 100 timesteps

*Original work due to Paul Crozier and Mark Stevens at Sandia National Labs

Simulation using AMD Firestream 9170
Molecular Dynamics: LAMMPS

GPU Acceleration

- Initialization
  - N_step / N_step_nn
  - NN Calc
    - N_step_nn
    - Stream Read pos, vel
    - Pair Potential
    - Stream Write
    - Propagator
    - Finalization

• Implementation uses Brook+
Molecular Dynamics: LAMMPS

Benchmark Tests

- **General:**
  - Single-core performance benchmarks
  - GPGPU implementation single-precision
  - 32,000 atoms, 100 timesteps (standard LAMMPS benchmark)

- **Test #1: GPGPU**
  - Pair Potential calc on GPGPU, full neighbor list, newton=off, no Coulomb table

- **Test #2: CPU ("identical" algorithm, identical model)**
  - Pair Potential calc on CPU, full neighbor list, newton=off, no Coulomb table

- **Test #3: CPU (optimized algorithm, identical model)**
  - Pair Potential calc on CPU, half neighbor list, newton=off, no Coulomb table

- **Test #4: CPU (optimized algorithm, optimized model)**
  - Pair Potential calc on CPU, half neighbor list, newton=on, Coulomb table

- **ASCI RED single-core performance (from LAMMPS website)**
  - Most likely a Test #4, included here for reference
Molecular Dynamics: LAMMPS
Rhodopsin Benchmark

Note: Early results (2008) using FireStream 9170 and ATI Stream SDK v1.1

Amadahl’s Law: Pair Potential compared with total time: 35% (Test#1), 75% (Test#2), 83% (Test#4)
Quantum Chemistry: Two-Electron Integrals

- One of the most common approaches in quantum chemical modeling employs gaussian basis sets to represent the electronic orbitals of the system.
- A computationally costly component of these calculations involves the evaluation of two-electron integrals.

\[
(\mu \nu | \lambda \tau) = \int \int dr_1 dr_2 \phi_\mu (r_1) \phi_\nu (r_2) \frac{1}{r_{12}} \phi_\lambda (r_1) \phi_\tau (r_2)
\]

\[
\phi_\mu (r) \sim \sum_k d_k g (\alpha_k , r) \\
g (\alpha_k , r) \sim \exp (-\alpha_k r^2)
\]

- For a gaussian basis, evaluation of two-electron integrals reduces to summation over closed-form expression (Boys, 1949).
- Features of expression required to be evaluated:
  - Certain pair quantities can be factored and pre-calculated
  - Expression contains +, -, *, /, sqrt(), exp(), erf()
Quantum Chemistry: Two-Electron Integrals

GPU Acceleration

- Implementation uses Brook+
Quantum Chemistry: Two-Electron Integrals

### STO-6G(1s) 4x4x4

<table>
<thead>
<tr>
<th>GPU Compute</th>
<th>GPU Setup</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATI/4870/single</td>
<td>0.678 sec</td>
<td>0.968 sec</td>
</tr>
<tr>
<td>AMD/9950(3GHz)/single</td>
<td>0.290 sec</td>
<td>2.262 sec</td>
</tr>
<tr>
<td>244x</td>
<td>814x</td>
<td></td>
</tr>
<tr>
<td>ATI/4870/double</td>
<td>0.241 sec</td>
<td>2.728 sec</td>
</tr>
<tr>
<td>AMD/9950(3GHz)/double</td>
<td>2.487 sec</td>
<td>198.749 sec</td>
</tr>
<tr>
<td>72x</td>
<td>80x</td>
<td></td>
</tr>
<tr>
<td>Nvidia/8800GTX/single*</td>
<td>1.123 sec</td>
<td></td>
</tr>
<tr>
<td>AMD/175/GAMESS*</td>
<td>90.6 sec</td>
<td></td>
</tr>
</tbody>
</table>

*Ufimtsev and Martinez (early work)

- Large number of integral limit (~10 million)
  - SP: **814x** speedup over CPU core
  - DP: **80x** speedup over CPU core
- CPU implementation definitely not optimized
- GPU performance/speedup will nevertheless be substantial

Note: Early results (2009) using Radeon HD 4870 and ATI Stream SDK v1.3
Software Design for Many-Core Processors

• Early work focused on GPU as an accelerator
• Identify likely algorithms to show acceleration, report speedup
• However, scientists and engineers do not choose their applications, applications are chosen by the problems we need to solve
• This inspired the code design project designated “Auburn Lake”
• Objective: explore software design for many-core processors
  • Key technology is OpenCL
  • OpenCL is NOT a GPGPU programming language
  • OpenCL is a device-independent API for fine-grained parallel programming
Software Design for GPGPU

GPU as an Accelerator

... Question – Is this really how we should use GPUs?

• Pros
  • Potential performance boost, easy way to get started (ok, maybe not so easy)
• Cons
  • Aamadahl’s Law will catch up with you
  • Overhead can be substantial (data transfer, data structure)
Software Design for GPGPU

GPU as a Processor?

- Different idea
  - Architect software for many-core GPGPU
  - Multi-core host acts as operating system
  - Same multi-core is “re-targeted” for compute

... Leads to another question, are we supposed to re-write our codes?
... No, seriously, the entire code?

Note CPU count: physical = 1, logical = 2
A Few Words About “Re-Writing” Our Codes

• Re-writing our codes at appropriate times is good
• No one actually re-writes code, coders will transform code
• Most production codes are selected for features unrelated to performance
  • Available documentation and examples
  • Usability and I/O
  • Robust model support, e.g., pseudo-potentials for a DFT code
• Code unrelated to performance does not need to be re-written
• Most large codes contain a multiplicity of variations on a theme
  • Re-write the theme, the variations follow with less effort
• We should not re-write code without a good reason
• When the architectures we run on change, that is a very good reason
• The history of HPC is filled with illusions of false permanence
• Source code should be living documents, not monuments to the programmer
Selecting a Terrible Calculation for GPU-Acceleration

• Classical MD with Stillinger-Weber potential for silicon

• Why is this a bad candidate for GPU Acceleration?
  • MD w/short-range classical potential dominated by nearest-neighbor bookkeeping
  • Stillinger-Weber potential for silicon:
    • Low FLOP complexity
    • Three-body force leads to unavoidable scatter writes
  • Sparse O(N) computation with data collision hazard

... Not your first choice for writing the typical “supercomputer in a box” paper.
Silicon Defect Diffusion (1/2)

• Molecular Dynamics can be used to study thermally induced infrequent events

• Examples include defect diffusion in solids, can be used to study growth of large defects impacting material properties

• Simulation is challenging
  • Requires small timestep to correctly model thermal motion
  • Requires long simulations to observe events

• For most of the simulation nothing actually happens – events are infrequent
Silicon defect diffusion (2/2)

- Transition events occur over relatively small time intervals relative to the overall simulation time.
- The real motion of interest is in the transition events.
- The thermal motion is necessary, but uninteresting.

![Graph showing local energy of selected atom over time](image-url)
Techniques for Accelerated Dynamics

- Parallel Replica – (Voter, 1998)
  - Replicas of the system are run independently on multiple processors
  - First transition on any processor halts all runs
  - Accumulated time from all processors reflects correct dynamics
- Depends critically on determining occurrence of transition & correct transition time

![Diagram showing the process of Accelerated Dynamics](image)

**N Replicas**

**Transition?**

**Re-crossing?**

**Replicate**

**De-phase**
Components of a Parallel Replica MD Code

- Potential – for silicon we can use a Stillinger-Weber three-body potential
- Propagators
  - Langevin – provides NVT ensemble
  - Parallel random number generator – LCG64 adapted from SPRNG
  - Velocity-Verlet – provides NVE for testing
- Transition detection
  - Steepest decent minimization (“stop and quench technique”)
  - Real-time multi-resolution analysis (use wavelets for detection)

... objective is to use many-core GPGPU processors to perform parallel replica MD
Unconventional Use of a Many-Core Processor
(Appears to be many, which one is the “core”?)

- View as 20 CPU “cores” with 16-way multi-threading and SSE for single-precision
  - For double-precision, no SSE
- Use the 20 “cores” to run parallel replica ensembles
  - SIMD engines are completely decoupled, which is the architectural reality anyway
- Not an obvious choice, not very stream-like
• CPU host acts as a scheduler, performs no computation
MD Without Nearest Neighbor Lists

• LAMMPS and nearest-neighbor (NN) lists revisited
  • Most MD codes are driven by NN list - $O(N^2)$ problem becomes $O(N)$ problem
  • Leads to arbitrary global memory access, not good for GPGPU
  • This is not the only approach

• Group atoms into small cells large enough to ensure that each atom can only interact with other atoms in adjacent cells
  • Still provides the pre-screening needed for an $O(n)$ algorithm

• For silicon the physics provides some order to this approach
  • Approximately 8 atoms per cell with NN cut-off less than cell size
Three-Body Potentials

- Stillinger-Weber potential has the form:

\[ v - \frac{1}{2} \sum_{ij} \phi(r_{ij}) + \sum_{ijk} g(r_{ij}) g(r_{ik}) \left( \cos \theta_{ijk} + \frac{1}{3} \right)^2 \]

- Two-body term leads to a simple optimization: \( f_{12} = -f_{21} \)
  - Can chose to exploit this, or not – factor of 2x in FLOPS
- Three-body term is more difficult: \( F_1 = f_{12} + f_{13}, F_2 = f_{12}, F_3 = f_{13} \)
  - Difficult to exploit this on GPGPU – leads to bad scatter memory write

- The issue of precision:
  - Forces \textit{calculated} with \textit{double-precision}
  - Positions, velocities and total forces \textit{stored} in \textit{single-precision}
Replica Replica

- Short-range radial cutoff imposes data collision limit

- One of 20 SIMD engines ("cores")

- For silicon, *approximately* 8 atoms per cell – dynamics changes number and cell association
- Small system, large replica count configuration - ~ 512 atoms, 80 replicas (using 20 “cores”)
- For large systems, fewer replicas, completely scalable solution
Hardware and Software Setup

- Hardware
  - ASUS M4A79T
  - Phenom X6 1090T BE @ 3.2 GHz ($300)
  - ATI Radeon HD 5970 ($700)
  - ATI Radeon HD 5870 ($400)
- Software
  - CentOS 5.4
  - GCC 4.1
  - ATI Stream SDK v2.1
  - Brown Deer COPRTHR SDK v1.1 (current)

“cygnus” hybrid CPU/GPU workstation
## Benchmarks

<table>
<thead>
<tr>
<th>Processor</th>
<th>Code/API</th>
<th>Time per Replica</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AMD Phenom II X6 1090T 3.2 GHz (CPU)</strong></td>
<td>LAMMPS/C++</td>
<td>0.58 sec</td>
<td>4.76 sec</td>
</tr>
<tr>
<td>AMD/ATI Radeon HD 5870 (GPU)</td>
<td>Auburn/OpenCL</td>
<td>0.29 sec</td>
<td>2.27 sec</td>
</tr>
<tr>
<td><strong>AMD Phenom II X6 1090T 3.2 GHz (CPU)</strong></td>
<td>Auburn/OpenCL</td>
<td>0.40 sec</td>
<td>3.35 sec</td>
</tr>
<tr>
<td><strong>AMD Phenom II X6 1090T 3.2 GHz (CPU)</strong></td>
<td>Auburn/OpenCL</td>
<td>0.44 sec</td>
<td>3.65 sec</td>
</tr>
</tbody>
</table>

1. Time per replica per 1000 steps
2. LAMMPS June 2010 reference code compiled with GCC 4.1
3. AMD OpenCL compiler and runtime provided by ATI Stream SDK v2.1
4. BDT OpenCL runtime provided by Brown Deer COPRTHR SDK v1.1
Real HPC on a Gamer Board

- 215 atom silicon vacancy diffusion
- *Entire simulation* performed on ATI Radeon HD 5870
- CPU host acts as a scheduler
- OpenCL - portable code for future processors
  - Same source - AMD CPU/GPU, NVIDIA GPU, Intel CPU
Conclusion: We have a Path Forward

• Typical question: what algorithms in MD code can be “accelerated” by GPU?
• Question now: what algorithms in MD code cannot run on the GPU?
• Design exercise challenges conventional approach to GPU as accelerator
• Confirms approach to software design for many-core (and multi-core)

Practical consequence: **ALL development** by Brown Deer will use OpenCL

  • Platform/vendor independent, broad industry support - future proof
  • One source, one design, run on anything in foreseeable future, efficiently
    • AMD CPU/GPU, NVIDIA GPU, Intel CPU, ...