Accelerating Computational Biology by 100x Using CUDA

John Stone
Theoretical and Computational Biophysics Group, University of Illinois
GPU Computing

- Commodity devices, omnipresent in modern computers
- Massively parallel hardware, hundreds of processing units, throughput oriented design
- Support all standard integer and floating point types
- Programming tools allow software to be written in dialects of familiar C/C++ and integrated into legacy software
- GPU algorithms are often multicore-friendly due to attention paid to data locality and work decomposition (e.g. MCUDA)
What Speedups Can GPUs Achieve?

• Single-GPU speedups of 8x to 30x vs. CPU core are quite common

• Best speedups (100x!) are attained on codes that are skewed towards floating point arithmetic, esp. CPU-unfriendly operations that prevent effective use of SSE or other vectorization

• Amdahl’s Law can prevent legacy codes from achieving peak speedups with only shallow GPU acceleration efforts
Computational Biology’s Insatiable Demand for Processing Power

- Simulations still fall short of biological timescales
- Large simulations extremely difficult to prepare, analyze
- Order of magnitude increase in performance would allow use of more sophisticated models
Fluorescence Microscopy

- 2-D reaction-diffusion simulation used to predict results of fluorescence microphotolysis experiments
- Simulate 1-10 second microscopy experiments, 0.1ms integration timesteps
- Goal: <= 1 min per simulation on commodity PC hardware
- Project home page: http://www.ks.uiuc.edu/Research/microscope/
Fluorescence Microscopy (2)

• Challenges for CPU:
  – Efficient handling of boundary conditions
  – Large number of floating point operations per timestep

• Challenges for GPU w/ CUDA:
  – Hiding global memory latency, improving memory access patterns, controlling register use
  – Few arithmetic operations per memory reference (for a GPU...)

Fluorescence Microscopy (3)

- Simulation runtime, software development time:
  - Original research code (CPU): 80 min
  - Optimized algorithm (CPU): 27 min
    - 40 hours of work
  - SSE-vectorized (CPU): 8 min
    - 20 hours of work
  - CUDA w/ 8800GTX: 38 sec, 12 times faster than SSE!
    - 12 hours of work, possible to improve further, but already “fast enough” for real use
    - CUDA code was more similar to the original than to the SSE vectorized version - arithmetic is almost “free” on the GPU
Calculating Electrostatic Potential Maps

- Used in molecular structure building, analysis, visualization, simulation
- Electrostatic potentials evaluated on a uniformly spaced 3-D lattice
- Each lattice point contains sum of electrostatic contributions of all atoms
Direct Coulomb Summation

- At each lattice point, sum potential contributions for all atoms in the simulated structure:
  - \( \text{potential}[j] += \frac{\text{charge}[i]}{R_{ij}} \)
Direct Coulomb Summation on the GPU

- GPU outruns a CPU core by 44x
- Work is decomposed into tens of thousands of independent threads, multiplexed onto hundreds of GPU processor cores
- Single-precision FP arithmetic is adequate for intended application
- Numerical accuracy can be further improved by compensated summation, spatially ordered summation groupings, or accumulation of potential in double-precision
- Starting point for more sophisticated algorithms
Direct Coulomb Summation on the GPU
Direct Coulomb Summation Runtime

Lower is better

Performance vs. Size

Potential lattice evaluation in seconds

Number of atoms

100
10
1
0.1
0.01
0.001
100
1000
10000
100000
1000000

direct summation, CPU

direct summation, 1 GPU

GPU underutilized

GPU fully utilized, ~40x faster than CPU

Accelerating molecular modeling applications with graphics processors.


Optimizing for the GPU

- Increase arithmetic intensity, reuse in-register data by “unrolling” lattice point computation into inner atom loop
- Each atom contributes to several lattice points, distances only differ in the X component:
  - potentialA += charge[i] / (distanceA to atom[i])
  - potentialB += charge[i] / (distanceB to atom[i]) ...
CUDA Block/Grid Decomposition

Unrolling increases computational tile size

Thread blocks: 64-256 threads

Threads compute up to 8 potentials. Skipping by half-warps optimizes global mem. perf.

Grid of thread blocks:

0,0 0,1 ...
1,0 1,1 ...
... ... ...

Padding waste
Direct Coulomb Summation Performance

**CUDA-Simple:** 14.8x faster, 33% of fastest GPU kernel

**CUDA-Unroll8clx:** fastest GPU kernel, 44x faster than CPU, 291 GFLOPS on GeForce 8800GTX

**CUDA-Simple Kernel**
**CUDA-Unroll4x Kernel**
**CUDA-Unroll8x Kernel**
**CUDA-Unroll8clx Kernel**
**CUDA-Unroll8csx Kernel**
**Intel QX6700 SSE3 Kernel**

Multi-GPU Direct Coulomb Summation

- Effective memory bandwidth scales with the number of GPUs utilized
- PCIe bus bandwidth not a bottleneck for this algorithm
- 117 billion evals/sec
- 863 GFLOPS
- 131x speedup vs. CPU core
- Power: 700 watts during benchmark

Quad-core Intel QX6700
Three NVIDIA GeForce 8800GTX
Multi-GPU Direct Coulomb Summation

- 4-GPU (2 Quadroplex) Opteron node at NCSA
- 157 billion evals/sec
- 1.16 TFLOPS
- 176x speedup vs. Intel QX6700 CPU core w/ SSE

- 4-GPU (GT200)
- 241 billion evals/sec
- 1.78 TFLOPS
- 271x speedup vs. Intel QX6700 CPU core w/ SSE
Cutoff Summation

• At each lattice point, sum potential contributions for atoms within cutoff radius:
  – if (distance to atom[i] < cutoff)
  – potential += (charge[i] / r) * s(r)
• Smoothing function s(r) is algorithm dependent
Infinite vs. Cutoff Potentials

• Infinite range potential:
  – All atoms contribute to all lattice points
  – Summation algorithm has quadratic complexity

• Cutoff (range-limited) potential:
  – Atoms contribute to lattice points within cutoff distance
  – Summation algorithm has linear time complexity
  – Has many applications in molecular modeling:
    • Replace electrostatic potential with shifted form
    • Short-range part for fast methods of approximating full electrostatics
    • Used for fast decaying interactions (e.g. Lennard-Jones, Buckingham)
Cutoff Summation on the GPU

Atoms spatially hashed into fixed-size “bins” in global memory

CPU handles bin overflows

- Atoms
  - Global memory
    - Bins of 8 atoms
    - Potential map regions
  - Shared memory
    - Atom bin

- Constant memory
  - Bin-Region neighborlist

- Process atom bins for current potential map region
Using the CPU to Improve GPU Performance

• GPU performs best when the work evenly divides into the number of threads/processing units

• Optimization strategy:
  – Use the CPU to “regularize” the GPU workload
  – Handle exceptional or irregular work units on the CPU while the GPU processes the bulk of the work
  – On average, the GPU is kept highly occupied, attaining a much higher fraction of peak performance
Cutoff Summation Runtime

NAMD Parallel Molecular Dynamics


- Designed from the beginning as a parallel program
- Uses the Charm++ philosophy:
  - Decompose computation into a large number of objects
  - Intelligent run-time system (Charm++) assigns objects to processors for dynamic load balancing with minimal communication

Hybrid of spatial and force decomposition:
- Spatial decomposition of atoms into cubes (called patches)
- For every pair of interacting patches, create one object for calculating electrostatic interactions
- Recent: Blue Matter, Desmond, etc. use this idea in some form
NAMD Overlapping Execution

Phillips et al., SC2002.

Example Configuration

847 objects

Angle Compute Objects

Offload to GPU

Multicast

100,000

Asynchronous Reductions

Point to Point

Patches: Integration

Objects are assigned to processors and queued as data arrives.
Nonbonded Forces on G80 GPU

- Start with most expensive calculation: direct nonbonded interactions.
- Decompose work into pairs of patches, identical to NAMD structure.
- GPU hardware assigns patch-pairs to multiprocessors dynamically.

Force computation on single multiprocessor (GeForce 8800 GTX has 16)

- Texture Unit
  - Force Table Interpolation
  - 8kB cache

- 16kB Shared Memory
  - Patch A Coordinates & Parameters

- 32-way SIMD Multiprocessor
  - 32-256 multiplexed threads
  - 32kB Registers
  - Patch B Coords, Params, & Forces

- Constants
  - Exclusions
  - 8kB cache

768 MB Main Memory, no cache, 300+ cycle latency

Nonbonded Forces CUDA Code

texture<float4> force_table;
__constant__ unsigned int exclusions[];
__shared__ atom jatom[];
atom iatom;       // per-thread atom, stored in registers
float4 iforce;    // per-thread force, stored in registers
for ( int j = 0; j < jatom_count; ++j ) {
    float dx = jatom[j].x - iatom.x; float dy = jatom[j].y - iatom.y; float dz = jatom[j].z - iatom.z;
    float r2 = dx*dx + dy*dy + dz*dz;
    if ( r2 < cutoff2 ) {
        float4 ft = texfetch(force_table, 1.f/sqrt(r2));
        bool excluded = false;
        int indexdiff = iatom.index - jatom[j].index;
        if ( abs(indexdiff) <= (int) jatom[j].excl_maxdiff ) {
            indexdiff += jatom[j].excl_index;
            excluded = ((exclusions[indexdiff>>5] & (1<<(indexdiff&31))) != 0);
        }
        float f = iatom.half_sigma + jatom[j].half_sigma; // sigma
        f *= f*f; // sigma^3
        f *= f; // sigma^6
        f *= ( f * ft.x + ft.y ); // sigma^12 * fi.x - sigma^6 * fi.y
        f *= iatom.sqrt_epsilon * jatom[j].sqrt_epsilon;
        float qq = iatom.charge * jatom[j].charge;
        if ( excluded ) { f = qq * ft.w; } // PME correction
        else { f += qq * ft.z; } // Coulomb
        iforce.x += dx * f; iforce.y += dy * f; iforce.z += dz * f;
        iforce.w += 1.f; // interaction count or energy
    }
}
GPU kernels are launched asynchronously, CPU continues with its own work, polling for GPU completion periodically. Forces needed by remote nodes are explicitly scheduled to be computed ASAP to improve overall performance.
Molecular Simulations: Virology

- Simulations lead to better understanding of the mechanics of viral infections
- Better understanding of infection mechanics at the molecular level may result in more effective treatments for diseases
- Since viruses are large, their computational “viewing” requires tremendous resources, in particular large parallel computers
- GPUs can significantly accelerate the simulation, analyses, and visualization of such structures
NAMD Performance on NCSA GPU Cluster, April 2008

<table>
<thead>
<tr>
<th>CPU Cores &amp; GPUs</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>60</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU-accelerated performance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Local blocks/GPU</td>
<td>13186</td>
<td>5798</td>
<td>2564</td>
<td>1174</td>
<td>577</td>
</tr>
<tr>
<td>Remote blocks/GPU</td>
<td>1644</td>
<td>1617</td>
<td>1144</td>
<td>680</td>
<td>411</td>
</tr>
<tr>
<td>GPU s/step</td>
<td>0.544</td>
<td>0.274</td>
<td>0.139</td>
<td>0.071</td>
<td>0.040</td>
</tr>
<tr>
<td>Total s/step</td>
<td>0.960</td>
<td>0.483</td>
<td>0.261</td>
<td>0.154</td>
<td>0.085</td>
</tr>
<tr>
<td>Unaccelerated performance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total s/step</td>
<td>6.76</td>
<td>3.33</td>
<td>1.737</td>
<td>0.980</td>
<td>0.471</td>
</tr>
<tr>
<td>Speedup from GPU acceleration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Factor</td>
<td>7.0</td>
<td>6.9</td>
<td>6.7</td>
<td>6.4</td>
<td>5.5</td>
</tr>
</tbody>
</table>

STMV benchmark, 1M atoms, 12A cutoff, PME every 4 steps, running on 2.4 GHz AMD Opteron + NVIDIA Quadro FX 5600
NAMD Performance on NCSA GPU Cluster, April 2008

- STMV virus (1M atoms)
- 60 GPUs match performance of 330 CPU cores
- 5.5-7x overall application speedup w/ G80-based GPUs
- Overlap with CPU
- Off-node results done first
- Plans for better performance
  - Tune or port remaining work
  - Balance GPU load

STMV Performance

2.4 GHz Opteron + Quadro FX 5600
NAMD Performance on GT200 GPU Cluster, August 2008

• 8 GT200s, 240 SPs @ 1.3GHz:
  – 72x faster than a single CPU core
  – 9x overall application speedup vs. 8 CPU cores
  – 32% faster overall than 8 nodes of G80 cluster
  – GT200 CUDA kernel is 54% faster
  – ~8% variation in GPU load

• Cost of double-precision for force accumulation is minimal: only 8% slower than single-precision

• LIVE DEMO on 4 GT200s in Exhibition Hall, NVIDIA booth 220
<table>
<thead>
<tr>
<th>Calculation / Algorithm</th>
<th>Algorithm class</th>
<th>Speedup vs. Intel QX6700 CPU core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fluorescence microphotolysis</td>
<td>Iterative matrix / stencil</td>
<td>12x</td>
</tr>
<tr>
<td>Pairlist calculation</td>
<td>Particle pair distance test</td>
<td>10-11x</td>
</tr>
<tr>
<td>Pairlist update</td>
<td>Particle pair distance test</td>
<td>5-15x</td>
</tr>
<tr>
<td>Molecular dynamics non-bonded force calc.</td>
<td>N-body cutoff force calculations</td>
<td>10x</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20x (w/ pairlist)</td>
</tr>
<tr>
<td>Cutoff electron density sum</td>
<td>Particle-grid w/ cutoff</td>
<td>15-23x</td>
</tr>
<tr>
<td>MSM short-range</td>
<td>Particle-grid w/ cutoff</td>
<td>24x</td>
</tr>
<tr>
<td>MSM long-range</td>
<td>Grid-grid w/ cutoff</td>
<td>22x</td>
</tr>
<tr>
<td>Direct Coulomb summation</td>
<td>Particle-grid</td>
<td>44x</td>
</tr>
</tbody>
</table>
Lessons Learned

• GPU algorithms need fine-grained parallelism and sufficient work to fully utilize the hardware

• Fine-grained GPU work decompositions compose well with the comparatively coarse-grained decompositions used for multicore or distributed memory programming

• Much of GPU algorithm optimization revolves around efficient use of multiple memory systems and latency hiding
Lessons Learned (2)

• The host CPU can potentially be used to “regularize” the computation for the GPU, yielding better overall performance

• Overlapping CPU work with GPU can hide some communication and unaccelerated computation
Ongoing and Future Work

• Visualization of multi-million atom biomolecular complexes
  – Migrate structural geometry and volumetric computations to the GPU
  – GPU accelerated ray tracing, ambient occlusion lighting, ...

• GPU acceleration of long running molecular dynamics trajectory analyses

• More opportunities available than time to pursue them!
Acknowledgements

- Prof. Klaus Schulten, David Hardy, Jim Phillips, Theoretical and Computational Biophysics Group, University of Illinois at Urbana-Champaign
- Prof. Wen-mei Hwu, Chris Rodrigues, John Stratton, IMPACT Group, University of Illinois at Urbana-Champaign
- The CUDA team at NVIDIA
- NVIDIA, NCSA (GPU clusters)
- NIH support: P41-RR05969
Publications

http://www.ks.uiuc.edu/Research/gpu/


