Experiences Developing and Maintaining Scientific Applications on GPU-Accelerated Platforms

John E. Stone
Theoretical and Computational Biophysics Group
Beckman Institute for Advanced Science and Technology
University of Illinois at Urbana-Champaign
http://www.ks.uiuc.edu/Research/gpu/
Big Red 2 Workshop
IUPUI Indianapolis, January 24, 2014
Goal: A Computational Microscope

Study the molecular machines in living cells

Ribosome: target for antibiotics

Poliovirus
NAMD and VMD Use GPUs & Petascale Computing to Meet Computational Biology’s Insatiable Demand for Processing Power

Number of atoms

- Lysozyme
- ApoA1
- ATP Synthase
- Ribosome
- STMV
- HIV capsid

Timeline from 1986 to 2014.
First Simulation of a Virus Capsid (2006)

Satellite Tobacco Mosaic Virus (STMV)

First MD simulation of a complete virus capsid
STMV smallest available capsid structure

STMV simulation, visualization, and analysis pushed us toward GPU computing!

MD showed that STMV capsid collapses without its RNA core

1 million atoms
A huge system for 2006

VMD – “Visual Molecular Dynamics”

- Visualization and analysis of:
  - molecular dynamics simulations
  - quantum chemistry calculations
  - particle systems and whole cells
  - sequence data

- User extensible w/ scripting and plugins

- http://www.ks.uiuc.edu/Research/vmd/

Electrons in Vibrating Buckyball

Poliovirus

Ribosome Sequences

Cellular Tomography, Cryo-electron Microscopy

Whole Cell Simulations
GPU Computing

• Commodity devices, omnipresent in modern computers (over a million sold per week)
• Massively parallel hardware, hundreds of processing units, throughput oriented architecture
• Standard integer and floating point types supported
• 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 data-parallel work decomposition
What Speedups Can GPUs Achieve?

- Single-GPU speedups of 10x to 30x vs. one CPU core are common.
- Best speedups can reach 100x or more, attained on codes dominated by floating point arithmetic, especially native GPU machine instructions, e.g. expf(), rsqrtf(), …
- Amdahl’s Law can prevent legacy codes from achieving peak speedups with shallow GPU acceleration efforts.
# CUDA GPU-Accelerated Trajectory Analysis and Visualization in VMD

<table>
<thead>
<tr>
<th>GPU-Accelerated Feature or Kernel</th>
<th>Typical speedup vs. a single CPU core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Molecular orbital display</td>
<td>120x</td>
</tr>
<tr>
<td>Radial distribution function</td>
<td>92x</td>
</tr>
<tr>
<td>Ray tracing w/ shadows</td>
<td>46x</td>
</tr>
<tr>
<td>Electrostatic field calculation</td>
<td>44x</td>
</tr>
<tr>
<td>Molecular surface display</td>
<td>40x</td>
</tr>
<tr>
<td>Ion placement</td>
<td>26x</td>
</tr>
<tr>
<td>MDFF density map synthesis</td>
<td>26x</td>
</tr>
<tr>
<td>Implicit ligand sampling</td>
<td>25x</td>
</tr>
<tr>
<td>Root mean squared fluctuation</td>
<td>25x</td>
</tr>
<tr>
<td>Radius of gyration</td>
<td>21x</td>
</tr>
<tr>
<td>Close contact determination</td>
<td>20x</td>
</tr>
<tr>
<td>Dipole moment calculation</td>
<td>15x</td>
</tr>
</tbody>
</table>
Peak Arithmetic Performance: Exponential Trend

Theoretical GFLOP/s

- NVIDIA GPU Single Precision
- NVIDIA GPU Double Precision
- Intel CPU Double Precision
- Intel CPU Single Precision

GeForce GTX TITAN
GeForce GTX 680
GeForce GTX 580
GeForce GTX 480
GeForce GTX 580 Ultra
GeForce 7800 GTX
Tesla M2090
Tesla C2050
Sandy Bridge
Woodcrest
Harpertown
Bloomfield
Westmere
Pontium 4
Apr-01 Sep-02 Jan-04 May-05 Oct-06 Feb-08 Jul-09 Nov-10 Apr-12 Aug-13 Dec-14
Comparison of CPU and GPU Hardware Architecture

**CPU**: Cache heavy, focused on individual thread performance

**GPU**: ALU heavy, massively parallel, throughput oriented
**NVIDIA Kepler GPU**

- ~3-6 GB DRAM Memory w/ ECC
- GPC GPC GPC GPC
- GPC GPC GPC GPC
- 1536KB Level 2 Cache

**Streaming Multiprocessor - SMX**

- 64 KB Constant Cache
- 64 KB L1 Cache / Shared Memory
- 48 KB Tex + Read-only Data Cache

**Graphics Processor Cluster**

**SMX SMX**

**Tex Unit**

16 × Execution block = 192 SP, 64 DP, 32 SFU, 32 LDST
What Runs on a GPU?

• GPUs run data-parallel programs called “kernels”

• GPUs are managed by a host CPU thread:
  – Create a CUDA context
  – Allocate/deallocate GPU memory
  – Copy data between host and GPU memory
  – Launch GPU kernels
  – Query GPU status
  – Handle runtime errors
CUDA Stream of Execution

- Host CPU thread launches a CUDA “kernel”, a memory copy, etc. on the GPU
- GPU action runs to completion
- Host synchronizes with completed GPU action
CUDA Grid/Block/Thread Decomposition

1-D, 2-D, or 3-D
Computational Domain

1-D, 2-D, or 3-D (SM >= 2.x)
Grid of thread blocks:

0,0 0,1 ...

1,0 1,1 ...

... ... ...

1-D, 2-D, 3-D thread block:

Padding arrays out to full blocks optimizes global memory performance by guaranteeing memory coalescing.
CUDA Work Abstractions: Grids, Thread Blocks, Threads

Thread blocks are scheduled onto pool of GPU SMs…

1-D, 2-D, 3-D (SM >= 2.x)

Grid of thread blocks:

SM / SMX

1-D, 2-D, 3-D thread block:
An Approach to Writing CUDA Kernels

• Find an algorithm that can expose substantial parallelism, we’ll ultimately need thousands of independent threads…

• Identify appropriate GPU memory or texture subsystems used to store data used by kernel

• Are there trade-offs that can be made to exchange computation for more parallelism?
  – Though counterintuitive, past successes resulted from this strategy
  – “Brute force” methods that expose significant parallelism do surprisingly well on GPUs

• Analyze the real-world use case for the problem and select a specialized kernel for the problem sizes that will be heavily used
GPUs Require ~20,000 Independent Threads for Full Utilization, Latency Hidding

Lower is better

Host thread
GPU Cold Start:
context init,
device binding,
kernel PTX JIT:
~110ms

GPU underutilized

GPU fully utilized,
~40x faster than CPU

Getting Performance From GPUs

• Don’t worry (much) about counting arithmetic operations…at least until you have nothing else left to do

• GPUs provide tremendous memory bandwidth, but even so, memory bandwidth often ends up being the performance limiter

• Keep/reuse data in registers as long as possible

• The main consideration when programming GPUs is accessing memory efficiently, and storing operands in the most appropriate memory system according to data size and access pattern
GPU Memory Systems

• GPU arithmetic rates dwarf memory bandwidth

• For Kepler K20 hardware:
  – ~2 TFLOPS vs. ~250 GB/sec
  – The ratio is roughly 40 FLOPS per memory reference for single-precision floating point

• GPUs include multiple fast on-chip memories to help narrow the gap:
  – Registers
  – Constant memory (64KB)
  – Shared memory (48KB / 16KB)
  – Read-only data cache / Texture cache (48KB)
Loop Unrolling, Register Tiling

…for (atomid=0; atomid<numatoms; atomid++) {
    float dy = coory - atominfo[atomid].y;
    float dysqpdzsq = (dy * dy) + atominfo[atomid].z;
    float x = atominfo[atomid].x;
    float dx1 = coorx1 - x;
    float dx2 = coorx2 - x;
    float dx3 = coorx3 - x;
    float dx4 = coorx4 - x;
    float charge = atominfo[atomid].w;
    energyvalx1 += charge * rsqrtf(dx1*dx1 + dysqpdzsq);
    energyvalx2 += charge * rsqrtf(dx2*dx2 + dysqpdzsq);
    energyvalx3 += charge * rsqrtf(dx3*dx3 + dysqpdzsq);
    energyvalx4 += charge * rsqrtf(dx4*dx4 + dysqpdzsq);
}
Avoid Output Conflicts, Conversion of Scatter to Gather

• Many CPU codes contain algorithms that “scatter” outputs to memory, to reduce arithmetic

• Scattered output can create bottlenecks for GPU performance due to bank conflicts

• On the GPU, it’s often better to do more arithmetic, in exchange for a regularized output pattern, or to convert “scatter” algorithms to “gather” approaches
Avoid Output Conflicts: Privatization Schemes

- **Privatization**: use of private work areas for workers
  - Avoid/reduce the need for thread synchronization barriers
  - Avoid/reduce the need atomic increment/decrement operations during work, use *parallel reduction* at the end…

- By working in separate memory buffers, workers *avoid read/modify/write conflicts* of various kinds

- Huge GPU thread counts make it impractical to privatize data on a per-thread basis, so GPUs must use *coarser granularity: warps, thread-blocks*

- Use of the *on-chip shared memory* local to each SM can often be considered a form of privatization
Example: avoiding output conflicts when summing numbers among threads in a block

Accumulate sums in thread-local registers before doing any reduction among threads

N-way output conflict:
Correct results require costly barrier synchronizations or atomic memory operations ON EVERY ADD to prevent threads from overwriting each other…

Parallel reduction: no output conflicts, Log2(N) barriers
Using the CPU to Optimize 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
  - Use fixed size bin data structures, with "empty" slots skipped or producing zeroed out results
  - Handle exceptional or irregular work units on the CPU; GPU processes the bulk of the work concurrently
  - On average, the GPU is kept highly occupied, attaining a high fraction of peak performance
Science 5: Quantum Chemistry Visualization

- Chemistry is the result of atoms sharing electrons
- Electrons occupy “clouds” in the space around atoms
- Calculations for visualizing these “clouds” are costly: tens to hundreds of seconds on CPUs – non-interactive
- GPUs enable the dynamics of electronic structures to be animated interactively for the first time

Taxol: cancer drug

VMD enables interactive display of QM simulations, e.g. Terachem, GAMESS
GPU Solution: Computing C\textsubscript{60} Molecular Orbitals

<table>
<thead>
<tr>
<th>Device</th>
<th>CPUs, GPUs</th>
<th>Runtime (s)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>2x Intel X5550-SSE</td>
<td>8</td>
<td>4.13</td>
<td>1</td>
</tr>
<tr>
<td>GeForce GTX 480</td>
<td>1</td>
<td>0.255</td>
<td>16</td>
</tr>
<tr>
<td>GeForce GTX 480</td>
<td>4</td>
<td>0.081</td>
<td>51</td>
</tr>
</tbody>
</table>

3-D orbital lattice: millions of points

D CUDA grid on one GPU

CUDA thread blocks

GPU threads each compute one point.

Lattice slices computed on multiple GPUs

NIH BTRC for Macromolecular Modeling and Bioinformatics
http://www.ks.uiuc.edu/
Writing SSE kernels for CPUs requires assembly language, compiler intrinsics, various libraries, or a really smart autovectorizing compiler and lots of luck...
Molecular Orbital Inner Loop in CUDA

for (shell=0; shell < maxshell; shell++) {
    float contracted_gto = 0.0f;
    for (prim=0; prim<num_prim_per_shell[shell_counter]; prim++) {
        float exponent = const_basis_array[prim_counter + prim_counter + 1];
        float contract_coeff = const_basis_array[prim_counter + prim_counter + 1];
        contracted_gto += contract_coeff * exp2f(-exponent*dist2);
        prim_counter += 2;
    }
    float tmpshell=0;
    switch (const_shell_symmetry[shell_counter]) {
        case S_SHELL:
            value += const_wave_f[ifunc++] * contracted_gto;    break;
        case P_SHELL:
            tmpshell += const_wave_f[ifunc++] * xdist;
            tmpshell += const_wave_f[ifunc++] * ydist
            tmpshell += const_wave_f[ifunc++] * zdist;
            value += tmpshell * contracted_gto;    break;
    }
    if (shell == maxshell) {
        return value;
    }
}

Aaaahhhhh….
Data-parallel CUDA kernel looks like normal C code for the most part…. 
NAMD Titan XK7 Performance August 2013

NAMD XK7 vs. XE6
Speedup: 3x-4x

HIV-1 Data: ~1.2 TB/day @ 4096 XK7 nodes
VMD Petascale Visualization and Analysis

• Analyze/visualize large trajectories too large to transfer off-site:
  – Compute time-averaged electrostatic fields, MDFF quality-of-fit, etc.
  – User-defined parallel analysis operations, data types
  – Parallel rendering, movie making

• Parallel I/O rates up to 275 GB/sec on 8192 Cray XE6 nodes – can read in 231 TB in 15 minutes!

• Multi-level dynamic load balancing tested with up to 262,144 CPU cores

• Supports GPU-accelerated Cray XK7 nodes for both visualization and analysis usage

NCSA Blue Waters Hybrid Cray XE6 / XK7 Supercomputer
  22,640 XE6 CPU nodes
  4,224 XK7 nodes w/ GPUs support fast VMD OpenGL movie rendering and visualization
VMD for Demanding Analysis Tasks
Parallel VMD Analysis w/ MPI

- Compute time-averaged electrostatic fields, MDFF quality-of-fit, etc.
- Parallel rendering, movie making
- User-defined parallel reduction operations, data types
- **Parallel I/O on Blue Waters:**
  - 109 GB/sec on 512 nodes
  - 275 GB/sec on 8,192 nodes
- **Timeline per-residue SASA calc. achieves 800x speedup @ 1000 BW XE6 nodes**
- Supports GPU-accelerated clusters and supercomputers
VMD as an Analysis Platform
Over 60 VMD Plugins Developed by Users

- VMD/NAMD sister programs, VMD is crucial for simulation analysis
- VMD user-extensible scripting w/ Tcl/Tk, Python
- Compiled C/C++ plugins loaded from shared libraries at runtime via dlopen()
- 70 molfile plugins provide access to molecular file formats
- **Built-in analysis commands exploit** XE6 multi-core CPUs, XK7Tesla K20X GPUs
- **New VMD collective ops and work scheduling interfaces enable existing code to be parallelized easily**
**Radial Distribution Function**

- RDFs describes how atom density varies with distance
- Can be compared with experiments
- Shape indicates phase of matter: sharp peaks appear for solids, smoother for liquids
- Normalized histogram of particle pair distances
Multi-GPU RDF Performance

- 4 NVIDIA GTX480 GPUs 30 to 92x faster than 4-core Intel X5550 CPU
- Fermi GPUs ~3x faster than GT200 GPUs: larger on-chip shared memory

Solid

Liquid

Time-Averaged Electrostatics Analysis on Energy-Efficient GPU Cluster

• **1.5 hour** job (CPUs) reduced to **3 min** (CPUs+GPU)
• Electrostatics of thousands of trajectory frames averaged
• Per-node power consumption on NCSA “AC” GPU cluster:
  – CPUs-only: 448 Watt-hours
  – CPUs+GPUs: 43 Watt-hours
• GPU Speedup: **25.5x**
• Power efficiency gain: **10.5x**

# Time-Averaged Electrostatics Analysis on NCSA Blue Waters

<table>
<thead>
<tr>
<th>NCSA Blue Waters Node Type</th>
<th>Seconds per trajectory frame for one compute node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cray XE6 Compute Node: 32 CPU cores (2xAMD 6200 CPUs)</td>
<td>9.33</td>
</tr>
<tr>
<td>Cray XK6 GPU-accelerated Compute Node: 16 CPU cores + NVIDIA X2090 (Fermi) GPU</td>
<td>2.25</td>
</tr>
</tbody>
</table>

**Speedup for GPU XK6 nodes vs. CPU XE6 nodes**

**Test Results:**

- XK6 nodes are 4.15x faster overall
- In progress…
- XK7 nodes 4.3x faster overall

**Tests on XK7 nodes indicate MSM is CPU-bound with the Kepler K20X GPU.**
- Performance is not much faster (yet) than Fermi X2090
- Need to move spatial hashing, prolongation, interpolation onto the GPU…

Preliminary performance for VMD time-averaged electrostatics w/ Multilevel Summation Method on the NCSA Blue Waters Early Science System
Multilevel Summation on the GPU

Accelerate **short-range cutoff** and **lattice cutoff** parts

Performance profile for 0.5 Å map of potential for 1.5 M atoms. Hardware platform is Intel QX6700 CPU and NVIDIA GTX 280.

<table>
<thead>
<tr>
<th>Computational steps</th>
<th>CPU (s)</th>
<th>w/ GPU (s)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short-range cutoff</td>
<td>480.07</td>
<td>14.87</td>
<td>32.3</td>
</tr>
<tr>
<td>Long-range anterpolation</td>
<td>0.18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>restriction</td>
<td>0.16</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>lattice cutoff</strong></td>
<td>49.47</td>
<td>1.36</td>
<td>36.4</td>
</tr>
<tr>
<td>prolongation</td>
<td>0.17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>interpolation</td>
<td>3.47</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>533.52</td>
<td>20.21</td>
<td>26.4</td>
</tr>
</tbody>
</table>

VMD “QuickSurf” Representation

- Displays continuum of structural detail:
  - All-atom models
  - Coarse-grained models
  - Cellular scale models
  - Multi-scale models: All-atom + CG, Brownian + Whole Cell
  - Smoothly variable between full detail, and reduced resolution representations of very large complexes

Fast Visualization of Gaussian Density Surfaces for Molecular Dynamics and Particle System Trajectories.

VMD “QuickSurf” Representation

- Uses multi-core CPUs and GPU acceleration to enable smooth real-time animation of MD trajectories
- Linear-time algorithm, scales to millions of particles, as limited by memory capacity
QuickSurf Algorithm Overview

- Build spatial acceleration data structures, optimize data for GPU
- Compute 3-D density map, 3-D volumetric texture map:
  \[ \rho(\vec{r}; \vec{r}_1, \vec{r}_2, \ldots, \vec{r}_N) = \sum_{i=1}^{N} e^{-\frac{|\vec{r} - \vec{r}_i|^2}{2\alpha^2}} \]
- Extract isosurface for a user-defined density value

3-D density map lattice, spatial acceleration grid, and extracted surface
QuickSurf Density Map Algorithm

- Spatial acceleration grid cells are sized to match the cutoff radius for the exponential, beyond which density contributions are negligible.
- Density map lattice points computed by summing density contributions from particles in 3x3x3 grid of neighboring spatial acceleration cells.
- Volumetric texture map is computed by summing particle colors normalized by their individual density contribution.

3-D density map lattice point and the neighboring spatial acceleration cells it references.
QuickSurf Density Parallel Decomposition

QuickSurf 3-D density map decomposes into thinner 3-D slabs/slices (CUDA grids)

Small 8x8 thread blocks afford large per-thread register count, shared memory

Each thread computes one or more density map lattice points

Padding optimizes global memory performance, guaranteeing coalesced global memory accesses

Grid of thread blocks

Threads producing results that are used

Inactive threads, region of discarded output

Large volume computed in multiple passes, or multiple GPUs

Chunk 0

Chunk 1

Chunk 2

QuickSurf Density Parallel Decomposition
Challenge: Support GPU-accelerated QuickSurf for **Large** Biomolecular Complexes

- Structures such as HIV initially needed all XK7 GPU memory to generate detailed surface renderings
- Goals and approach:
  - **Avoid slow CPU-fallback!**
  - Incrementally change algorithm phases to use more compact data types, while maintaining performance
  - Specialize code for different precision/performance/memory capacity cases
Supporting Multiple Data Types for QuickSurf Density Maps and Marching Cubes Vertex Arrays

• The major algorithm components of QuickSurf are now used for many other purposes:
  – Gaussian density map algorithm now used for MDFF Cryo EM density map fitting methods in addition to QuickSurf
  – Marching Cubes routines also used for Quantum Chemistry visualizations of molecular orbitals

• Rather than simply changing QuickSurf to use a particular internal numerical representation, it is desirable to instead use CUDA C++ templates to make type-generic versions of the key objects, kernels, and output vertex arrays

• Accuracy-sensitive algorithms use high-precision data types, performance and memory capacity sensitive cases use quantized or reduced precision approaches
Minimizing the Impact of Generality on QuickSurf Code Complexity

• A critical factor in the simplicity of supporting multiple QuickSurf data types arises from the so-called “gather” oriented algorithm we employ
  – Internally, all in-register arithmetic is single-precision
  – Compressed or reduced precision data type conversions are performed on-the-fly as needed
• Small inlined type conversion routines are defined for each of the cases we want to support
• Key QuickSurf kernels made type-generic using C++ template syntax, and the compiler automatically generates type-specific kernels as needed
Example Templated Density Map Kernel

template<class DENSITY, class VOLTEX>
__global__ static void
gaussdensity_fast_tex_norm(int natoms,
    const float4 * RESTRICT sorted_xyzr,
    const float4 * RESTRICT sorted_color,
    int3 numvoxels,
    int3 acncells,
    float acgridspacing,
    float invacgridspacing,
    const uint2 * RESTRICT cellStartEnd,
    float gridspacing, unsigned int z,
    DENSITY * RESTRICT densitygrid,
    VOLTEX * RESTRICT voltexmap,
    float invisovalue) {

Example Templated Density Map Kernel

template<class DENSITY, class VOLTEX>
__global__ static void
gaussdensity_fast_tex_norm( ... ) {

  ... Triple-nested and unrolled inner loops here ...

  DENSITY densityout;
  VOLTEX texout;
  convert_density(densityout, densityval1);
  densitygrid[outaddr] = densityout;
  convert_color(texout, densitycol1);
  voltexmap[outaddr] = texout;
Net Result of QuickSurf Memory Efficiency Optimizations

• **Halved** overall GPU memory use

• **Achieved 1.5x to 2x performance gain:**
  – The “gather” density map algorithm keeps type conversion operations out of the innermost loop
  – Density map global memory writes reduced to half
  – Multiple stages of Marching Cubes operate on smaller input and output data types
  – Same code path supports multiple precisions

• **Users now get full GPU-accelerated QuickSurf in many cases that previously triggered CPU-fallback, all platforms (laptop/desk/super) benefit!**
VMD “QuickSurf” Representation, Ray Tracing

All-atom HIV capsid simulations w/ up to 64M atoms on Blue Waters
Ray Tracing Molecular Graphics

- Ambient occlusion lighting, shadows, reflections, transparency, and more…
- Satellite tobacco mosaic virus capsid w/ ~75K atoms
Lighting Comparison

Two lights, no shadows

Two lights, hard shadows, 1 shadow ray per light

Ambient occlusion + two lights, 144 AO rays/hit
BW VMD/Tachyon Movie Generation

chromatophore from purple bacteria
200 proteins, 3700 cofactors
10 million atoms

480 XE6 nodes for 85m @ 4096x2400
BW VMD/Tachyon Movie Generation

20 M atom chromatophore patch

360 XE6 nodes for 3h50m @ 4096x2400
GPU Ray Tracing of HIV-1 on Blue Waters

- Ambient occlusion lighting, shadows, transparency, antialiasing, depth cueing, **144 rays/pixel minimum**
- 64 million atom virus simulation
- 1000+ movie frames
- Surface generation and ray tracing stages each use $\geq 75\%$ of GPU memory
VMD GPU Ray Tracing of HIV-1 Capsid
HIV-1 Parallel HD Movie Rendering on Blue Waters Cray XE6/XK7

New “TachyonL-OptiX” on XK7 vs. Tachyon on XE6: K20X GPUs yield **up to eight times** geom+ray tracing speedup

<table>
<thead>
<tr>
<th>Node Type and Count</th>
<th>Script Load Time</th>
<th>State Load Time</th>
<th>Geometry + Ray Tracing</th>
<th>Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>256 XE6 CPUs</td>
<td>7 s</td>
<td>160 s</td>
<td>1,374 s</td>
<td>1,541 s</td>
</tr>
<tr>
<td>512 XE6 CPUs</td>
<td>13 s</td>
<td>211 s</td>
<td>808 s</td>
<td>1,032 s</td>
</tr>
<tr>
<td>64 XK7 Tesla K20X GPUs</td>
<td>2 s</td>
<td>38 s</td>
<td>655 s</td>
<td>695 s</td>
</tr>
<tr>
<td>128 XK7 Tesla K20X GPUs</td>
<td>4 s</td>
<td>74 s</td>
<td>331 s</td>
<td>410 s</td>
</tr>
<tr>
<td>256 XK7 Tesla K20X GPUs</td>
<td>7 s</td>
<td>110 s</td>
<td><strong>171 s</strong></td>
<td><strong>288 s</strong></td>
</tr>
</tbody>
</table>

Acknowledgements

• Theoretical and Computational Biophysics Group, University of Illinois at Urbana-Champaign
• NCSA Blue Waters Team
• NVIDIA CUDA Center of Excellence, University of Illinois at Urbana-Champaign
• Many of the staff at NVIDIA and Cray
• Funding:
  – NSF OCI 07-25070
  – NSF PRAC “The Computational Microscope”
  – NIH support: 9P41GM104601, 5R01GM098243-02
GPU Computing Publications
http://www.ks.uiuc.edu/Research/gpu/


GPU Computing Publications
http://www.ks.uiuc.edu/Research/gpu/


GPU Computing Publications
http://www.ks.uiuc.edu/Research/gpu/

- **Adapting a message-driven parallel application to GPU-accelerated clusters.**

- **GPU acceleration of cutoff pair potentials for molecular modeling applications.**

