GPU Computing:
Applications, Libraries and Programming
## Molecular Dynamics (MD) Applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Features Supported</th>
<th>GPU Perf</th>
<th>Release Status</th>
<th>Notes/Benchmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMBER</td>
<td>PMEMD Explicit Solvent &amp; GB Implicit Solvent</td>
<td>&gt; 100 ns/day JAC NVE on 2X K20s</td>
<td>Released Multi-GPU, multi-node</td>
<td>AMBER 12, GPU Revision Support 12.1 <a href="http://ambermd.org/gpus/benchmarks.htm#Benchmarks">http://ambermd.org/gpus/benchmarks.htm#Benchmarks</a></td>
</tr>
<tr>
<td>CHARMM</td>
<td>Implicit (5x), Explicit (2x) Solvent via OpenMM</td>
<td>2x C2070 equals 32-35x X5667 CPUs</td>
<td>Released Single &amp; multi-GPU in single node</td>
<td>Release C37b1; <a href="http://www.charmm.org/news/c37b1.html#postjump">http://www.charmm.org/news/c37b1.html#postjump</a></td>
</tr>
<tr>
<td>DL_POLY</td>
<td>Two-body Forces, Link-cell Pairs, Ewald SPME forces, Shake VV</td>
<td>4x</td>
<td>Released V 4.03, Multi-GPU, multi-node supported</td>
<td>Source only, Results Published <a href="http://www.stfc.ac.uk/CSE/randd/ccg/software/DL_POLY/25526.aspx">http://www.stfc.ac.uk/CSE/randd/ccg/software/DL_POLY/25526.aspx</a></td>
</tr>
<tr>
<td>GROMACS</td>
<td>Implicit (5x), Explicit (2x) Solvent via OpenMM</td>
<td>165 ns/Day DHFR on 4X C2075s</td>
<td>Released (4.5 Single GPU Pre-beta (4.6 Multi-GPU)</td>
<td>Release 4.6 is 1st Multi-GPU support Q4 2012 Release</td>
</tr>
<tr>
<td>NAMD</td>
<td>Full electrostatics with PME and most simulation features</td>
<td>4.0 ns/days F1-ATPase on 1x K20X</td>
<td>Released 100M atom capable Multi-GPU, multi-node</td>
<td>NAMD 2.9</td>
</tr>
</tbody>
</table>

**Notes/ Benchmarks**
- GPU Perf benchmarked on GPU supported features and may be a kernel to kernel perf comparison.
- AMBER 12, GPU Revision Support 12.1 [http://ambermd.org/gpus/benchmarks.htm#Benchmarks](http://ambermd.org/gpus/benchmarks.htm#Benchmarks)
- Release C37b1; [http://www.charmm.org/news/c37b1.html#postjump](http://www.charmm.org/news/c37b1.html#postjump)
- Source only, Results Published [http://www.stfc.ac.uk/CSE/randd/ccg/software/DL_POLY/25526.aspx](http://www.stfc.ac.uk/CSE/randd/ccg/software/DL_POLY/25526.aspx)
The blue node contains Dual E5-2687W CPUs (8 Cores per CPU).

The green nodes contain Dual E5-2687W CPUs (8 Cores per CPU) and 1 or 2 NVIDIA K10, K20, or K20X GPUs.

Average speedup calculated from 4 AMBER, 3 NAMD, 3 LAMMPS, and 1 GROMACS test cases. Error bars show the maximum and minimum speedup for each hardware configuration.
K20 Accelerates Simulations of All Sizes

Gain 28x throughput/performance by adding just one K20 GPU when compared to dual CPU performance

Running AMBER 12 GPU Support Revision 12.1
SPFP with CUDA 4.2.9 ECC Off

The blue node contains 2x Intel E5-2687W CPUs (8 Cores per CPU)

Each green node contains 2x Intel E5-2687W CPUs (8 Cores per CPU) plus 1x NVIDIA K20 GPUs

Nucleosome
Outstanding Strong Scaling with Multi-STMV

100 STMV on Hundreds of Nodes

- **Fermi XK6**
- **CPU XK6**

Running NAMD version 2.9
Each **blue XE6 CPU node** contains 1x AMD 1600 Opteron (16 Cores per CPU).
Each **green XK6 CPU+GPU node** contains 1x AMD 1600 Opteron (16 Cores per CPU) and an additional 1x NVIDIA X2090 GPU.

Accelerate your science by **2.7-3.8x** when compared to CPU-based supercomputers
<table>
<thead>
<tr>
<th>Application</th>
<th>Features Supported</th>
<th>GPU Perf</th>
<th>Release Status</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abalone</td>
<td>Simulations (on 1060 GPU)</td>
<td>4-29X (on 1060 GPU)</td>
<td>Released, Version 1.8.51</td>
<td>Agile Molecule, Inc.</td>
</tr>
<tr>
<td>Ascalaph</td>
<td>Computation of non-valent interactions</td>
<td>4-29X (on 1060 GPU)</td>
<td>Released, Version 1.1.4</td>
<td>Agile Molecule, Inc.</td>
</tr>
<tr>
<td>ACEMD</td>
<td>Written for use only on GPUs</td>
<td>150 ns/day DHFR on 1x K20</td>
<td>Released; Single and multi-GPUs</td>
<td>Production bio-molecular dynamics (MD) software specially optimized to run on GPUs</td>
</tr>
<tr>
<td>Folding@Home</td>
<td>Powerful distributed computing molecular dynamics system; implicit solvent and folding</td>
<td>Depends upon number of GPUs</td>
<td>Released; GPUs and CPUs</td>
<td><a href="http://folding.stanford.edu">http://folding.stanford.edu</a></td>
</tr>
<tr>
<td>GPUGrid.net</td>
<td>High-performance all-atom biomolecular simulations; explicit solvent and binding</td>
<td>Depends upon number of GPUs</td>
<td>Released; NVIDIA GPUs only</td>
<td><a href="http://www.gpugrid.net/">http://www.gpugrid.net/</a></td>
</tr>
<tr>
<td>HALMD</td>
<td>Simple fluids and binary mixtures (pair potentials, high-precision NVE and NVT, dynamic correlations)</td>
<td>Up to 66x on 2090 vs. 1 CPU core</td>
<td>Released, Version 0.2.0</td>
<td><a href="http://halmd.org/benchmarks.html#supercool-ed-binary-mixture-kob-andersen">http://halmd.org/benchmarks.html#supercool-ed-binary-mixture-kob-andersen</a></td>
</tr>
<tr>
<td>HOOMD-Blue</td>
<td>Written for use only on GPUs</td>
<td>Kepler 2X faster than Fermi</td>
<td>Released, Version 0.11.1</td>
<td><a href="http://codeblue.umich.edu/hoomd-blue/">http://codeblue.umich.edu/hoomd-blue/</a></td>
</tr>
<tr>
<td>OpenMM</td>
<td>Implicit and explicit solvent, custom forces</td>
<td>Implicit: 127-213 ns/day Explicit: 18-55 ns/day DHFR</td>
<td>Released Version 4.1.1</td>
<td>Library and application for molecular dynamics on high-performance</td>
</tr>
</tbody>
</table>

GPU Perf compared against Multi-core x86 CPU socket.
GPU Perf benchmarked on GPU supported features and may be a kernel to kernel perf comparison.
# Quantum Chemistry Applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Features Supported</th>
<th>GPU Perf</th>
<th>Release Status</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abinit</td>
<td>Local Hamiltonian, non-local Hamiltonian, LOBPCG algorithm, diagonalization / orthogonalization</td>
<td>1.3-2.7X</td>
<td>Released since Version 6.12 Multi-GPU support</td>
<td><a href="http://www.abinit.org">www.abinit.org</a></td>
</tr>
<tr>
<td>ADF</td>
<td>Fock Matrix, Hessians</td>
<td>TBD</td>
<td>Pilot project completed, Under development Multi-GPU support</td>
<td><a href="http://www.scm.com">www.scm.com</a></td>
</tr>
<tr>
<td>Casino</td>
<td>TBD</td>
<td>TBD</td>
<td>Under development Spring 2013 release Multi-GPU support</td>
<td><a href="http://www.tcm.phy.cam.ac.uk/~mdt26/casino.html">http://www.tcm.phy.cam.ac.uk/~mdt26/casino.html</a></td>
</tr>
<tr>
<td>GAMESS-US</td>
<td>Libqc with Rys Quadrature Algorithm, Hartree-Fock, MP2 and CCSD in Q4 2012</td>
<td>1.3-1.6X, 2.3-2.9x HF</td>
<td>Released Multi-GPU support</td>
<td>Next release Q4 2012 <a href="http://www.msg.ameslab.gov/gamess/index.html">http://www.msg.ameslab.gov/gamess/index.html</a></td>
</tr>
</tbody>
</table>

**Notes:**
- GPU Perf compared against multi-core x86 CPU socket.
- GPU Perf benchmarked on GPU supported features and may be a kernel to kernel perf comparison.
<table>
<thead>
<tr>
<th>Application</th>
<th>Features Supported</th>
<th>GPU Perf</th>
<th>Release Status</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAMESS-UK</td>
<td>(ss</td>
<td>ss) type integrals within calculations using Hartree Fock \textit{ab initio} methods and density functional theory. Supports organics &amp; inorganics.</td>
<td>8x</td>
<td>Release in 2012 Multi-GPU support</td>
</tr>
<tr>
<td>GPAW</td>
<td>Electrostatic poisson equation, orthonormalizing of vectors, residual minimization method (rmm-diis)</td>
<td>8x</td>
<td>Released Multi-GPU support</td>
<td><a href="http://wiki.fysik.dtu.dk/gpaw/devel/projects/gpu.html">http://wiki.fysik.dtu.dk/gpaw/devel/projects/gpu.html</a>, Samuli Hakala (CSC Finland) &amp; Chris O’Grady (SLAC)</td>
</tr>
<tr>
<td>MOLCAS</td>
<td>CU.BLAS support</td>
<td>1.1x</td>
<td>Released, Version 7.8 Single GPU. Additional GPU support coming in Version 8</td>
<td><a href="http://www.molcas.org">www.molcas.org</a></td>
</tr>
<tr>
<td>MOLPRO</td>
<td>Density-fitted MP2 (DF-MP2), density fitted local correlation methods (DF-RHF, DF-KS), DFT</td>
<td>1.7-2.3X projected</td>
<td>Under development Multiple GPU</td>
<td><a href="http://www.molpro.net">www.molpro.net</a>, Hans-Joachim Werner</td>
</tr>
</tbody>
</table>

GPU Perf compared against Multi-core x86 CPU socket. GPU Perf benchmarked on GPU supported features and may be a kernel to kernel perf comparison.
# Quantum Chemistry Applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Features Supported</th>
<th>GPU Perf</th>
<th>Release Status</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOPAC2009</td>
<td>pseudodiagonalization, full diagonalization, and density matrix assembling</td>
<td>3.8-14X</td>
<td>Under Development Single GPU</td>
<td>Academic port. <a href="http://openmopac.net">http://openmopac.net</a></td>
</tr>
<tr>
<td>Octopus</td>
<td>DFT and TDDFT</td>
<td>TBD</td>
<td>Released</td>
<td><a href="http://www.tddft.org/programs/octopus/">http://www.tddft.org/programs/octopus/</a></td>
</tr>
<tr>
<td>PEtot</td>
<td>Density functional theory (DFT) plane wave pseudopotential calculations</td>
<td>6-10X</td>
<td>Released Multi-GPU</td>
<td>First principles materials code that computes the behavior of the electron structures of materials</td>
</tr>
</tbody>
</table>

GPU Perf compared against Multi-core x86 CPU socket. GPU Perf benchmarked on GPU supported features and may be a kernel to kernel perf comparison.
## Quantum Chemistry Applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Features Supported</th>
<th>GPU Perf</th>
<th>Release Status</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>QMCPACK</td>
<td>Main features</td>
<td>3-4x</td>
<td>Released Multiple GPUs</td>
<td>NCSA University of Illinois at Urbana-Champaign</td>
</tr>
<tr>
<td>Quantum Espresso/PWscf</td>
<td>PWscf package: linear algebra (matrix multiply), explicit computational kernels, 3D FFTs</td>
<td>2.5-3.5x</td>
<td>Released Version 5.0 Multiple GPUs</td>
<td>Created by Irish Centre for High-End Computing</td>
</tr>
</tbody>
</table>

GPU Perf compared against Multi-core x86 CPU socket. GPU Perf benchmarked on GPU supported features and may be a kernel to kernel perf comparison.
TeraChem running on 8 C2050s on 1 node
NWChem running on 4096 Quad Core CPUs In the Chinook Supercomputer
Giant Fullerene C240 Molecule

Similar performance from just a handful of GPUs
<table>
<thead>
<tr>
<th>Related Applications</th>
<th>Features Supported</th>
<th>GPU Perf</th>
<th>Release Status</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amira 5®</td>
<td>3D visualization of volumetric data and surfaces</td>
<td>70x</td>
<td>Released, Version 5.3.3 Single GPU</td>
<td>Visualization from Visage Imaging. Next release, 5.4, will use GPU for general purpose processing in some functions <a href="http://www.visageimaging.com/overview.html">http://www.visageimaging.com/overview.html</a></td>
</tr>
<tr>
<td>BINDSURF</td>
<td>Allows fast processing of large ligand databases</td>
<td>100X</td>
<td>Available upon request to authors; single GPU</td>
<td>High-Throughput parallel blind Virtual Screening, <a href="http://www.biomedcentral.com/1471-2105/13/S14/S13">http://www.biomedcentral.com/1471-2105/13/S14/S13</a></td>
</tr>
<tr>
<td>BUDE</td>
<td>Empirical Free Energy Forcefield</td>
<td>6.5-13.4X</td>
<td>Released Single GPU</td>
<td>University of Bristol <a href="http://www.bris.ac.uk/biochemistry/cpfg/bude/bude.htm">http://www.bris.ac.uk/biochemistry/cpfg/bude/bude.htm</a></td>
</tr>
<tr>
<td>PyMol</td>
<td>Lines: 460% increase Cartoons: 1246% increase Surface: 1746% increase Spheres: 753% increase Ribbon: 426% increase</td>
<td>1700x</td>
<td>Released, Version 1.5 Single GPUs</td>
<td><a href="http://pymol.org/">http://pymol.org/</a></td>
</tr>
<tr>
<td>VMD</td>
<td>High quality rendering, large structures (100 million atoms), analysis and visualization tasks, multiple GPU support for display of molecular orbitals</td>
<td>100-125X or greater on kernels</td>
<td>Released, Version 1.9</td>
<td>Visualization from University of Illinois at Urbana-Champaign <a href="http://www.ks.uiuc.edu/Research/vmd/">http://www.ks.uiuc.edu/Research/vmd/</a></td>
</tr>
<tr>
<td>Application</td>
<td>Features Supported</td>
<td>GPU Speedup</td>
<td>Release Status</td>
<td>Website</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------------------------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>BarraCUDA</td>
<td>Alignment of short sequencing reads</td>
<td>6-10x</td>
<td>Version 0.6.2 - 3/2012 Multi-GPU, multi-node</td>
<td><a href="http://seqbarracuda.sourceforge.net/">http://seqbarracuda.sourceforge.net/</a></td>
</tr>
<tr>
<td>CUDASW++</td>
<td>Parallel search of Smith-Waterman database</td>
<td>10-50x</td>
<td>Version 2.0.8 - Q1/2012 Multi-GPU, multi-node</td>
<td><a href="http://sourceforge.net/projects/cudasw/">http://sourceforge.net/projects/cudasw/</a></td>
</tr>
<tr>
<td>CUSHAW</td>
<td>Parallel, accurate long read aligner for large genomes</td>
<td>10x</td>
<td>Version 1.0.40 - 6/2012 Multiple-GPU</td>
<td><a href="http://cushaw.sourceforge.net/">http://cushaw.sourceforge.net/</a></td>
</tr>
<tr>
<td>GPU-BLAST</td>
<td>Protein alignment according to BLASTP</td>
<td>3-4x</td>
<td>Version 2.2.26 - 3/2012 Single GPU</td>
<td><a href="http://eudoxus.cheme.cmu.edu/gpublast/gpu">http://eudoxus.cheme.cmu.edu/gpublast/gpu</a> blast.html</td>
</tr>
<tr>
<td>GPU-HMMER</td>
<td>Parallel local and global search of Hidden Markov Models</td>
<td>60-100x</td>
<td>Version 2.3.2 - Q1/2012 Multi-GPU, multi-node</td>
<td><a href="http://www.mpihmmer.org/installguideGPUHMMER.htm">http://www.mpihmmer.org/installguideGPUHMMER.htm</a></td>
</tr>
<tr>
<td>mCUDA-MEME</td>
<td>Scalable motif discovery algorithm based on MEME</td>
<td>4-10x</td>
<td>Version 3.0.12 Multi-GPU, multi-node</td>
<td><a href="https://sites.google.com/site/yongchaosoftware/memduda-meme">https://sites.google.com/site/yongchaosoftware/memduda-meme</a></td>
</tr>
<tr>
<td>SeqNFind</td>
<td>Hardware and software for reference assembly, blast, SW, HMM, de novo assembly</td>
<td>400x</td>
<td>Released. Multi-GPU, multi-node</td>
<td><a href="http://www.seqnfind.com/">http://www.seqnfind.com/</a></td>
</tr>
<tr>
<td>UGENE</td>
<td>Fast short read alignment</td>
<td>6-8x</td>
<td>Version 1.11 - 5/2012 Multi-GPU, multi-node</td>
<td><a href="http://ugene.unipro.ru/">http://ugene.unipro.ru/</a></td>
</tr>
<tr>
<td>WideLM</td>
<td>Parallel linear regression on multiple similarly-shaped models</td>
<td>150x</td>
<td>Version 0.1-1 - 3/2012 Multi-GPU, multi-node</td>
<td><a href="http://insilicos.com/products/widelm">http://insilicos.com/products/widelm</a></td>
</tr>
</tbody>
</table>

GPU Perf compared against same or similar code running on single CPU machine
Performance measured internally or independently
## Physics Applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Features Supported</th>
<th>GPU Perf</th>
<th>Release Status</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chroma</td>
<td>Wilson-clover fermions, Krylov solvers, Domain-decomposition</td>
<td>Up to 9x 768 GPUs vs. 768 (16-core) CPUs</td>
<td>Multi-GPU support November 2009 Available now</td>
<td><a href="http://usqcd.jlab.org/usqcd-docs/chroma/">http://usqcd.jlab.org/usqcd-docs/chroma/</a></td>
</tr>
<tr>
<td>GTC</td>
<td>Electron push and shift (accounting for &gt;80% of run time)</td>
<td>2-3x 128 GPUs vs. 128 (16-core) CPUs</td>
<td>Multi-GPU support Available now</td>
<td>Simulates microturbulence &amp; transport in magnetically confined fusion plasma</td>
</tr>
<tr>
<td>QUDA</td>
<td>Wilson, Wilson-clover, Twisted mass, Improved staggered (asqtad or HISQ) and Domain wall</td>
<td>Up to 9x at various scales</td>
<td>Available Multi GPU</td>
<td>(URL??????)</td>
</tr>
</tbody>
</table>
## Numerical Analytics Applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Features Supported</th>
<th>GPU Perf</th>
<th>Release Status</th>
<th>Notes</th>
</tr>
</thead>
</table>
| MATLAB MathWorks     | • Over 200 GPU enabled MATLAB functions  
                       • CUDA kernels with MATLAB  
                       • Distributing standalone GPU applications with MATLAB | 2-20x    | Released since 2010    | Multi GPU                       |
|                      |                                                                                   |          |                        | Article Webinar Documentation  |
| Jacket for MATLAB    | • Several hundred GPU enabled MATLAB functions  
                       • CUDA kernels with MATLAB  
                       • Distributing standalone GPU applications with MATLAB | 2-20x    | Released since 2008    | Multi GPU                       |
| AccelerEyes          |                                                                                   |          |                        | Products Tour Documentation    |
| Mathematica Wolfram  | • GPU enabled Mathematica functions  
                       • Development environment for CUDA                                              | 2-20x    | Released since 2010    | Multi GPU                       |
|                      |                                                                                   |          |                        | Product Description User Guide|
|                      |                                                                                   |          |                        | Seminar                         |

GPU Performance compared against quad-core x86 CPU socket
GPU-Accelerated MATLAB Results

- **10x speedup** in data clustering via K-means clustering algorithm
- **14x speedup** in template matching routine (part of cancer cell image analysis)
- **3x speedup** in estimating 7.6 million contract prices using Black-Scholes model
- **17x speedup** in simulating the movement of 3072 celestial objects
- **4x speedup** in adaptive filtering routine (part of acoustic tracking algorithm)
- **4x speedup** in wave equation solving (part of seismic data processing algorithm)
<table>
<thead>
<tr>
<th>Application</th>
<th>GPU Features</th>
<th>GPU Perf</th>
<th>Production Status</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>COSMO</td>
<td>Entire model</td>
<td>3xDynamics</td>
<td>In development</td>
<td>Multi-GPU</td>
</tr>
<tr>
<td>WRF</td>
<td>WSM5, WSM3, Ice Microphysics models</td>
<td>4x-6x models</td>
<td>Today, release 3.2</td>
<td>single-GPU</td>
</tr>
<tr>
<td>ASUCA (JMA)</td>
<td>Entire model</td>
<td>12x total</td>
<td>Evaluation at JMA</td>
<td>multi-GPU</td>
</tr>
<tr>
<td>NIM</td>
<td>Dynamical core</td>
<td>7x dynamics</td>
<td>Under development</td>
<td>multi-GPU</td>
</tr>
<tr>
<td>HIRLAM</td>
<td>Dynamical core</td>
<td>3x dynamics</td>
<td>Demonstration</td>
<td>multi-GPU</td>
</tr>
<tr>
<td>HOMME</td>
<td>Dynamical core</td>
<td>3x kernels</td>
<td>Under development</td>
<td>single-GPU</td>
</tr>
<tr>
<td>CAM/SE</td>
<td>Entire model</td>
<td>2x dynamics</td>
<td>Under development</td>
<td>Multi-GPU</td>
</tr>
<tr>
<td>GEOS-5</td>
<td>Entire model</td>
<td>10x models, 3x dynamics</td>
<td>Demonstration</td>
<td>multi-GPU</td>
</tr>
<tr>
<td>MITgcm</td>
<td>Dynamical core</td>
<td>3x dynamics</td>
<td>Demonstration</td>
<td>single-GPU</td>
</tr>
<tr>
<td>HYCOM</td>
<td>Dynamical core</td>
<td>2x kernels</td>
<td>Demonstration</td>
<td>single-GPU</td>
</tr>
</tbody>
</table>

GPU Perf compared against Multi-core x86 CPU socket, features and may be a kernel to kernel perf comparison.
<table>
<thead>
<tr>
<th>Application</th>
<th>GPU Features</th>
<th>GPU Perf</th>
<th>Release Status</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Altair AcuSolve</td>
<td>Linear eqn solver</td>
<td>2x Total</td>
<td>Today, release 1.8a</td>
<td>FE unstructured NS, multi-GPU</td>
</tr>
<tr>
<td>ANSYS Fluent</td>
<td>Radiation heat transfer model</td>
<td>10x RHT Model, 2x AMG solver (beta)</td>
<td>Today, release 14.5</td>
<td>Multi-GPU RHT model, Single-GPU Solver</td>
</tr>
<tr>
<td>Autodesk Moldflow</td>
<td>Linear eqn solver</td>
<td>1.5x</td>
<td>Today, release 2013</td>
<td>FE unstructured NS, single-GPU</td>
</tr>
<tr>
<td>FluiDyna Culises- OpenFOAM</td>
<td>Linear eqn solvers</td>
<td>3x Solver</td>
<td>Today, release 1.2</td>
<td>Unstructured NS, single-GPU</td>
</tr>
<tr>
<td>FluiDyna LBultra</td>
<td>LBM, particle CFD</td>
<td>20x Total</td>
<td>Today, release 2.0</td>
<td>Structured LBM, multi-GPU</td>
</tr>
<tr>
<td>Vratis SpeedIT- OpenFOAM Solver</td>
<td>Linear eqn solvers</td>
<td>6x Solver</td>
<td>Today, release 1.2</td>
<td>Unstructured NS, multi-GPU</td>
</tr>
<tr>
<td>Vratis ARAEL</td>
<td>Linear eqn solvers</td>
<td>3x Solver</td>
<td>Today, release 1.0</td>
<td>Single-GPU</td>
</tr>
<tr>
<td>Prometech Particleworks</td>
<td>MPS, particle CFD</td>
<td>4x-9x Total</td>
<td>Today, release 3.0</td>
<td>Particle based, multi-GPU</td>
</tr>
<tr>
<td>Sandia NL S3D</td>
<td>Chemistry kernel</td>
<td>8x SP, 5x DP kernel</td>
<td>Demonstration</td>
<td>Structured DNS, multi-GPU</td>
</tr>
<tr>
<td>SD++ (SU - Jameson)</td>
<td>Explicit solver</td>
<td>15x Total</td>
<td>In development</td>
<td>FE unstructured NS, multi-GPU</td>
</tr>
<tr>
<td>FEFLO (GMU - Lohner)</td>
<td>Explicit solver</td>
<td>2-10x Total</td>
<td>In development</td>
<td>FE unstructured NS, multi-GPU</td>
</tr>
<tr>
<td>Turbostream</td>
<td>Explicit solver</td>
<td>19x Total</td>
<td>Today, release 2.0</td>
<td>Structured grid NS, multi-GPU</td>
</tr>
</tbody>
</table>

GPU Perf compared against Multi-core x86 CPU socket, features and may be a kernel to kernel perf comparison.
## Computational Structural Mechanics Applications

<table>
<thead>
<tr>
<th>Application</th>
<th>GPU Features</th>
<th>GPU Perf</th>
<th>Release Status</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS Mechanical</td>
<td>Linear eqn solvers</td>
<td>1.5-2.0x Total (1GPU over hex core)</td>
<td>Today, release 14.5</td>
<td>FE implicit, multi-GPU</td>
</tr>
<tr>
<td>Abaqus/Standard</td>
<td>Linear eqn solver</td>
<td>1.5-2.5x Total (1GPU over quad core)</td>
<td>Today, release 6.12</td>
<td>FE implicit, multi-GPU</td>
</tr>
<tr>
<td>IMPETUS Afea</td>
<td>Explicit solver, SPH</td>
<td>10x SPH, 2x Total</td>
<td>Today, release 1.0</td>
<td>FE explicit, multi-GPU</td>
</tr>
<tr>
<td>LS-DYNA Implicit</td>
<td>Linear eqn solver</td>
<td>3x Total</td>
<td>Beta available for CentOS Linux</td>
<td>FE implicit, multi-GPU</td>
</tr>
<tr>
<td>MSC Nastran</td>
<td>Linear eqn solver</td>
<td>1.5-2.5x Total (1GPU over quad core)</td>
<td>Today, release 2013 Alpha</td>
<td>FE implicit, multi-GPU</td>
</tr>
<tr>
<td>Marc</td>
<td>Linear eqn solver</td>
<td>1.5x Total</td>
<td>Today, release 2012</td>
<td>FE implicit, multi-GPU</td>
</tr>
<tr>
<td>RADIOSS Implicit</td>
<td>Linear eqn solver</td>
<td>2x-4x Total</td>
<td>Today, release 12</td>
<td>FE implicit, multi-GPU</td>
</tr>
<tr>
<td>PAM-CRASH Implicit</td>
<td>Linear eqn solver</td>
<td>1.5x Total</td>
<td>Planned, 2012</td>
<td>FE implicit, single-GPU</td>
</tr>
<tr>
<td>NX Nastran</td>
<td>Linear eqn solver</td>
<td>1.4x Total</td>
<td>Planned, 2013</td>
<td>Single-GPU</td>
</tr>
</tbody>
</table>

GPU Perf compared against Multi-core x86 CPU socket.
GPU Perf benchmarked on GPU supported features and may be a kernel to kernel perf comparison.
3 Ways to Accelerate Applications

- Libraries: “Drop-in” Acceleration
- OpenACC Directives: Easily Accelerate Applications
- Programming Languages: Maximum Flexibility
NVIDIA CUDA Library Approach

- Provide basic building blocks
- Make them easy to use
- Make them fast

- Provides a quick path to GPU acceleration
- Enables developers to focus on their “secret sauce”
- Ideal for applications that use CPU libraries
GPU Accelerated Libraries

“Drop-in” Acceleration for Your Applications
cuBLAS Level 3: >1 TFLOPS double-precision

![GFLOPS and Speedup over MKL graphs]

- MKL 10.3.6 on Intel SandyBridge E5-2687W @3.10GHz
- CUBLAS 5.0.30 on K20X, input and output data on device
CuSPARSE Performance

CSRMV*

* Average of single, double, complex and double complex

- MKL 10.3.6 on Intel SandyBridge E5-2687W @3.10GHz
- CUBLAS 5.0.30 on K20X, input and output data on device
CUFFT: up to 600 GFLOPS

1D used in audio processing and as a foundation for 2D and 3D FFTs

- CUFFT 5.0.30 on K20X, input and output data on device
Thrust: STL-like CUDA Template Library

- GPU(device) and CPU(host) vector class
  ```
  thrust::host_vector<float> H(10, 1.f);
  thrust::device_vector<float> D = H;
  ```

- Iterators
  ```
  thrust::fill(D.begin(), D.begin()+5, 42.f);
  float* raw_ptr = thrust::raw_pointer_cast(D);
  ```

- Algorithms
  - Sort, reduce, transformation, scan, ..
    ```
    thrust::transform(D1.begin(), D1.end(), D2.begin(), D2.end(),
    thrust::plus<float>()); // D2 = D1 + D2
    ```

C++ STL Features for CUDA
Explore the CUDA (Libraries) Ecosystem

- CUDA Tools and Ecosystem described in detail on NVIDIA Developer Zone: developer.nvidia.com/

- Get a taste of CUDA libs
  - Test drive your app on the new K20

- See the demos
  - CUDA/CUDA library experts on call

© NVIDIA Corporation 2012
Summary

- CUDA libraries offer high performance for minimal effort
- Robust community of 3rd party libraries
- Familiar interfaces make porting legacy code easy (“drop-in”)
- Enables focus on core IP
3 Ways to Accelerate Applications

Applications

Libraries

“Drop-in” Acceleration

OpenACC Directives

Easily Accelerate Applications

Programming Languages

Maximum Flexibility
OpenACC Directives

Your original Fortran or C code

Simple Compiler hints

Compiler Parallelizes code

Works on many-core GPUs & multicore CPUs

Program myscience
... serial code ...
!
acc

kernels

do k = 1,n1

do i = 1,n2
... parallel code ...

dendo
dendo
!
acc end kernels
... 
End Program myscience
main() {
    double pi = 0.0; long i;
    #pragma omp parallel for reduction(+:pi)
    for (i=0; i<N; i++)
    {
        double t = (double)((i+0.05)/N);
        pi += 4.0/(1.0+t*t);
    }
    printf("pi = %f\n", pi/N);
}

main() {
    double pi = 0.0; long i;
    #pragma acc kernels
    for (i=0; i<N; i++)
    {
        double t = (double)((i+0.05)/N);
        pi += 4.0/(1.0+t*t);
    }
    printf("pi = %f\n", pi/N);
}
Directives: Easy & Powerful

Real-Time Object Detection
Global Manufacturer of Navigation Systems

Valuation of Stock Portfolios using Monte Carlo
Global Technology Consulting Company

Interaction of Solvents and Biomolecules
University of Texas at San Antonio

5x in 40 Hours

2x in 4 Hours

5x in 8 Hours

“Optimizing code with directives is quite easy, especially compared to CPU threads or writing CUDA kernels. The most important thing is avoiding restructuring of existing code for production applications.”

-- Developer at the Global Manufacturer of Navigation Systems
Focus on Exposing Parallelism

With Directives, tuning work focuses on exposing parallelism, which makes codes inherently better.

Example: Application tuning work using directives for new Titan system at ORNL

S3D
Research more efficient combustion with next-generation fuels

- Tuning top 3 kernels (90% of runtime)
- 3 to 6x faster on CPU+GPU vs. CPU+CPU
- But also improved all-CPU version by 50%

CAM-SE
Answer questions about specific climate change adaptation and mitigation scenarios

- Tuning top key kernel (50% of runtime)
- 6.5x faster on CPU+GPU vs. CPU+CPU
- Improved performance of CPU version by 100%
OpenACC Specification and Website

- Full OpenACC 1.0 Specification available online
  
  http://www.openacc-standard.org

- Quick reference card also available

- Implementations available now from PGI, Cray, and CAPS
A Very Simple Exercise: SAXPY

SAXPY in C

```c
void saxpy(int n,
    float a,
    float *x,
    float *restrict y)
{
    #pragma acc kernels
    for (int i = 0; i < n; ++i)
    y[i] = a*x[i] + y[i];
}
...
// Perform SAXPY on 1M elements
saxpy(1<<20, 2.0, x, y);
...
```

SAXPY in Fortran

```fortran
subroutine saxpy(n, a, x, y)
    real :: x(:), y(:), a
    integer :: n, i
$!acc kernels
    do i=1,n
        y(i) = a*x(i)+y(i)
    enddo
$!acc end kernels
end subroutine saxpy
...
$ Perform SAXPY on 1M elements
call saxpy(2**20, 2.0, x_d, y_d)
...
Directive Syntax

- **Fortran**
  
  
  `!$acc directive [clause [,] clause] ...`

  Often paired with a matching end directive surrounding a structured code block

  `!$acc end directive`

- **C**
  
  
  `#pragma acc directive [clause [,] clause] ...`

  Often followed by a structured code block
$\textit{kernels: Your first OpenACC Directive}$

Each loop executed as a separate \textit{kernel} on the GPU.

\begin{verbatim}
!$acc kernels
  do i=1,n
    a(i) = 0.0
    b(i) = 1.0
    c(i) = 2.0
  end do

  do i=1,n
    a(i) = b(i) + c(i)
  end do

!$acc end kernels
\end{verbatim}

Kernel:
A parallel function that runs on the GPU
Kernels Construct

Fortran

\begin{verbatim}
!$acc kernels [clause ...]
  structured block
!$acc end kernels
\end{verbatim}

C

\begin{verbatim}
#pragma acc kernels [clause ...]
  \{ structured block \}
\end{verbatim}

Clauses

\begin{verbatim}
if( condition )
async( expression )
\end{verbatim}

Also, any data clause (more later)
Trivial first example

- Apply a loop directive
- Learn compiler commands

```
#include <stdlib.h>

void saxpy(int n, float a, float *x, float *restrict y)
{
    #pragma acc kernels
    for (int i = 0; i < n; ++i)
    {
        y[i] = a * x[i] + y[i];
    }
}

int main(int argc, char **argv)
{
    int N = 1<<20; // 1 million floats
    if (argc > 1)
    {
        N = atoi(argv[1]);
    }
    float *x = (float*)malloc(N * sizeof(float));
    float *y = (float*)malloc(N * sizeof(float));
    for (int i = 0; i < N; ++i) {
        x[i] = 2.0f;
        y[i] = 1.0f;
    }
    saxpy(N, 3.0f, x, y);
    return 0;
}

*restrict: “I promise y does not alias x”
```
Compile and run

C:

pgcc -acc -ta=nvidia -Minfo=accel -o saxpy_acc saxpy.c

Fortran:

pgf90 -acc -ta=nvidia -Minfo=accel -o saxpy_acc saxpy.f90

Compiler output:

pgcc -acc -Minfo=accel -ta=nvidia -o saxpy_acc saxpy.c

saxpy:
8, Generating copyin(x[:n-1])
Generating copy(y[:n-1])
Generating compute capability 1.0 binary
Generating compute capability 2.0 binary
9, Loop is parallelizable
Accelerator kernel generated
9, #pragma acc loop worker, vector(256) /* blockIdx.x threadIdx.x */
  CC 1.0 : 4 registers; 52 shared, 4 constant, 0 local memory bytes; 100% occupancy
  CC 2.0 : 8 registers; 4 shared, 64 constant, 0 local memory bytes; 100% occupancy
3 Ways to Accelerate Applications

- Libraries
  - “Drop-in” Acceleration

- OpenACC Directives
  - Easily Accelerate Applications

- Programming Languages
  - Maximum Flexibility
Anatomy of a CUDA C/C++ Application

- **Serial** code executes in a **Host** (CPU) thread
- **Parallel** code executes in many **Device** (GPU) threads across multiple processing elements
void serial_function(...) {
    ...
}
void other_function(int ...) {
    ...
}
void saxpy_serial(float ...) {
    for (int i = 0; i < n; ++i)
        y[i] = a*x[i] + y[i];
}
void main( ) {
    float x;
    saxpy_serial(..);
    ...
}
CUDA C: C with a few keywords

void saxpy_serial(int n, float a, float *x, float *y) {
    for (int i = 0; i < n; ++i)
        y[i] = a*x[i] + y[i];
} // Invoke serial SAXPY kernel
saxpy_serial(n, 2.0, x, y);

__global__ void saxpy_parallel(int n, float a, float *x, float *y) {
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n) y[i] = a*x[i] + y[i];
} // Invoke parallel SAXPY kernel with 256 threads/block
int nbblocks = (n + 255) / 256;
saxpy_parallel<<<nbblocks, 256>>>(n, 2.0, x, y);
CUDA Kernels

- Parallel portion of application: execute as a kernel
  - Entire GPU executes kernel, many threads

- CUDA threads:
  - Lightweight
  - Fast switching
  - 1000s execute simultaneously

<table>
<thead>
<tr>
<th></th>
<th>Host</th>
<th>Executors functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPU</td>
<td>Device</td>
<td>Executes kernels</td>
</tr>
</tbody>
</table>
CUDA Kernels: Parallel Threads

- A **kernel** is a function executed on the GPU as an array of threads in parallel.

- All threads execute the same code, can take different paths.

- Each thread has an ID:
  - Select input/output data
  - Control decisions

```c
float x = input[threadIdx.x];
float y = func(x);
output[threadIdx.x] = y;
```
CUDA Kernels: Subdivide into Blocks
CUDA Kernels: Subdivide into Blocks

Threads are grouped into **blocks**
CUDA Kernels: Subdivide into Blocks

- Threads are grouped into **blocks**
- **Blocks** are grouped into a grid
CUDA Kernels: Subdivide into Blocks

- Threads are grouped into blocks
- Blocks are grouped into a grid
- A kernel is executed as a grid of blocks of threads
CUDA Kernels: Subdivide into Blocks

- Threads are grouped into **blocks**
- Blocks are grouped into a **grid**
- A **kernel** is executed as a **grid of blocks of threads**
Kernel Execution

- Each kernel is executed on one device
- Multiple kernels can execute on a device at one time

CUDA - enabled GPU

- Each thread is executed by a core
- Each block is executed by one SM and does not migrate
- Several concurrent blocks can reside on one SM depending on the blocks’ memory requirements and the SM’s memory resources
- Each kernel is executed on one device
- Multiple kernels can execute on a device at one time
Thread blocks allow cooperation

- Threads may need to cooperate:
  - Cooperatively load/store blocks of memory all will use
  - Share results with each other or cooperate to produce a single result
  - Synchronize with each other
Thread blocks allow scalability

- Blocks can execute in any order, concurrently or sequentially
- This independence between blocks gives scalability:
  - A kernel scales across any number of SMs
Rich Toolchain & Ecosystem for Fast Ramp-up on GPUs

- **Debuggers & Profilers**
  - cuda-gdb
  - NV Visual Profiler
  - Parallel Nsight
  - Visual Studio
  - Allinea
  - TotalView

- **GPU Compilers**
  - C
  - C++
  - Fortran
  - OpenCL
  - DirectCompute
  - Java
  - Python

- **Parallelizing Compilers**
  - PGI Accelerator
  - CAPS HMPP
  - mCUDA
  - OpenMP

- **Numerical Packages**
  - MATLAB
  - Mathematica
  - NI LabView
  - pyCUDA

- **Libraries**
  - BLAS
  - FFT
  - LAPACK
  - NPP
  - Sparse
  - Imaging
  - RNG

- **GPGPU Consultants & Training**
  - ANEO
  - GPU Tech

- **OEM Solution Providers**
  - Dell
  - HP
  - IBM
  - CRAY
  - ASUS
  - Supermicro
  - SGI
  - Fujitsu
  - Bull
  - Lenovo
  - NEC
Summary

- There’s an app for that!...maybe
  - www.nvidia.com/teslaapps
- Libraries
  - Drop-in acceleration.
  - Same as the familiar CPU libraries we already use.
- OpenACC
  - Small code changes and little effort with immediate results.
- CUDA C/C++
  - Best choice for highest-performance code.
Thank you!