The Next Generation of High Performance Computing

William Gropp

www.cs.illinois.edu/~wgropp
Extrapolation is Risky

- **1989 – T – 23 years**
  - Intel introduces 486DX
  - Eugene Brooks writes “Attack of the Killer Micros”
  - 4 years *before* TOP500
  - Top systems at about 2 GF Peak

- **1999 – T – 13 years**
  - NVIDIA introduces its GPU (GeForce 256)
    - Programming GPUs still a challenge 13 years later
  - Top system – ASCI Red, 9632 cores, 3.2 TF Peak (about 3 GPUs in 2012)
  - MPI is 7 years old
HPC Today

- High(est)-End systems
  - 1 PF ($10^{15}$ Ops/s) achieved on a few “peak friendly” applications
  - Much worry about scalability, how we’re going to get to an ExaFLOPS
  - Systems are all oversubscribed
    - DOE INCITE awarded almost 900M processor hours in 2009; 1600M-1700M hours in 2010-2012; (big jump planned in 2013 – over 5B hours)
    - NSF PRAC awards for Blue Waters similarly competitive

- Widespread use of clusters, many with accelerators; cloud computing services
  - These are transforming the low and midrange

- Laptops (far) more powerful than the supercomputers I used as a graduate student
HPC in 2012

- **Sustained** PF systems
  - K Computer (Fujitsu) at RIKEN, Kobe, Japan (2011)
  - “Sequoia” Blue Gene/Q at LLNL
  - NSF Track 1 “Blue Waters” at Illinois
  - Undoubtedly others (China, ...)

- Still programmed with MPI and MPI+other (e.g., MPI+OpenMP or MPI+OpenCL/CUDA)
  - But in many cases using toolkits, libraries, and other approaches
    - And not so bad – applications will be able to run when the system is turned on
    - Replacing MPI will require some compromise – e.g., domain specific (higher-level but less general)
      - Lots of evidence that fully automatic solutions won’t work
HPC in 2020-2023

- Exascale systems are likely to have
  - Extreme power constraints, leading to
    - Clock Rates similar to today’s systems
    - A wide-diversity of simple computing elements (simple for hardware but complex for software)
    - Memory per core and per FLOP will be much smaller
    - Moving data anywhere will be expensive (time and power)
  - Faults that will need to be detected and managed
    - Some detection may be the job of the programmer, as hardware detection takes power
  - Extreme scalability and performance irregularity
    - Performance will require enormous concurrency
    - Performance is likely to be variable
      - Simple, static decompositions will not scale
  - A need for latency tolerant algorithms and programming
    - Memory, processors will be 100s to 10000s of cycles away. Waiting for operations to complete will cripple performance
What Do Current Systems Tell Us?

• Examples of trends
  ♦ Supercomputers: Blue Waters
  ♦ Exploiting Commodity Computing: GPU Clusters
  ♦ Post GPU: Radical architectures

• Parallelism is about getting *performance*
  ♦ Productivity is important, but only if performance is achieved
  ♦ All systems already “heterogeneous”
    • “Vector” instructions really a separate unit
  ♦ **Sustained performance is the goal**
Focus on Sustained Performance

- **Blue Water’s and NSF are focusing on **sustained** performance in a way few have been before.**

- *Sustained* is the computer’s performance on a broad range of applications that scientists and engineers use every day.
  - Time to solution is the metric – not Ops/s
  - Tests include time to read data and write the results

- NSF’s call emphasized sustained performance, demonstrated on a collection of application benchmarks (application + problem set)
  - Not just simplistic metrics (e.g. HP Linpack)
  - Applications include both Petascale applications (effectively use the full machine, solving scalability problems for both compute and I/O) and applications that use a fraction of the system

- Blue Waters project focus is on delivering sustained PetaFLOPS performance to all applications
  - Develop tools, techniques, samples, that exploit all parts of the system
  - Explore new tools, programming models, and libraries to help applications get the most from the system
# Blue Waters Science Team Characteristics

<table>
<thead>
<tr>
<th>Science Area</th>
<th>Number of Teams</th>
<th>Codes</th>
<th>Structured Grids</th>
<th>Unstructured Grids</th>
<th>Dense Matrix</th>
<th>Sparse Matrix</th>
<th>N-Body</th>
<th>Monte Carlo</th>
<th>FFT</th>
<th>Significant I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>Climate and Weather</td>
<td>3</td>
<td>CESM, GCRM, CM1, HOMME</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plasmas/ Magnetosphere</td>
<td>2</td>
<td>H3D(M), OSIRIS, Magtail/ UPIC</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stellar Atmospheres and Supernovae</td>
<td>2</td>
<td>PPM, MAESTRO, CASTRO, SEDONA</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cosmology</td>
<td>2</td>
<td>Enzo, pGADGET</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Combustion/ Turbulence</td>
<td>1</td>
<td>PSDNS</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>General Relativity</td>
<td>2</td>
<td>Cactus, Harm3D, LazEV</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Molecular Dynamics</td>
<td>4</td>
<td>AMBER, Gromacs, NAMD, LAMMPS</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quantum Chemistry</td>
<td>2</td>
<td>SIAL, GAMESS, NWChem</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Material Science</td>
<td>3</td>
<td>NEMOS, OMEN, GW, QMCPACK</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Earthquakes/ Seismology</td>
<td>2</td>
<td>AWP-ODC, HERCULES, PLSQR, SPECFEM3D</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quantum Chromo Dynamics</td>
<td>1</td>
<td>Chroma, MILC, USQCD</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Social Networks</td>
<td>1</td>
<td>EPISIMDEmICS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Evolution</td>
<td>1</td>
<td>Eve</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Computer Science</td>
<td>1</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Heart of Blue Waters: Two New Chips

**AMD Interlagos**
*157 GF peak performance*

**Features:**
- 2.3-2.6 GHz
- 8 core modules, 16 threads
- On-chip Caches
  - L1 (I:8x64KB; D:16x16KB)
  - L2 (8x2MB)
- Memory Subsystem
  - Four memory channels
  - 51.2 GB/s bandwidth

**NVIDIA Kepler**
*1,400 GF peak performance*

**Features:**
- 15 Streaming multiprocessors (SMX)
  - SMX: 192 sp CUDA cores, 64 dp units, 32 special function units
  - L1 caches/shared mem (64KB, 48KB)
  - L2 cache (1536KB)
- Memory subsystem
  - Six memory channels
  - 180 GB/s bandwidth
Cray XE6 Nodes

- **Dual-socket Node**
  - Two AMD Interlagos chips
    - 16 core modules, 64 threads
    - 313 GFs peak performance
    - 64 GBs memory
      - 102 GB/sec memory bandwidth
  - Gemini Interconnect
    - Router chip & network interface
    - Injection Bandwidth (peak)
      - 9.6 GB/sec per direction

Blue Waters contains 22,640 Cray XE6 compute nodes.
Blue Waters contains 3,072 Cray XK7 compute nodes.

Cray XK7 Nodes

- **Dual-socket Node**
  - One AMD Interlagos chip
    - 32 GBs memory
    - 51.2 GB/s bandwidth
  - One NVIDIA Kepler chip
    - 1.4 TFs peak performance
    - 6 GBs GDDR5 memory
    - 180 GB/sec bandwidth
- **Gemini Interconnect**
  - Same as XE6 nodes
Gemini Interconnect Network

Blue Waters
3D Torus Size
23 x 24 x 24

Service Nodes spread throughout the torus
Blue Waters Disk Subsystem

- Cray Sonexion 1600
  - Lustre file system
  - Reliable, Modular, Scalable
  - Fully integrated
    - Servers
    - Disk drives (Scalable Storage Units)
    - QDR Infiniband switches
  - Hierarchical monitoring

- Blue Waters Disk Subsystem
  - Capacity: 34.6 PBs (raw), 25.9 PBs (usable)
  - Bandwidth: >1 TB/s (sustained)
Blue Waters Archive System

• Spectra Logic T-Finity
  ◦ Dual-arm robotic tape libraries
  ◦ High availability and reliability, with built-in redundancy

• Blue Waters Archive
  ◦ Capacity: 380 PBs (raw), 300 PBs (usable)
  ◦ Bandwidth: 100 GB/sec (sustained)
  ◦ RAIT for increased reliability
Blue Waters Computing System

- Blue Waters Computing System
  - Sonexion: 26 PBs
  - >1 TB/sec
  - 100 GB/sec
  - 10/40/100 Gb Ethernet Switch
- Spectra Logic: 300 PBs
  - 120+ Gb/sec
- Sonexion: 26 PBs
  - >1 TB/sec
- WAN
  - 100 GB/sec
  - IB Switch
## Blue Waters and Titan Computing Systems

<table>
<thead>
<tr>
<th>System Attribute</th>
<th>Blue Waters</th>
<th>NCSA</th>
<th>ORNL Titan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vendors</td>
<td>Cray/AMD/NVIDIA</td>
<td>Cray/AMD/NVIDIA</td>
<td></td>
</tr>
<tr>
<td>Processors</td>
<td>Interlagos/Kepler</td>
<td>Interlagos/Kepler</td>
<td></td>
</tr>
<tr>
<td>Total Peak Performance (PF)</td>
<td>11.9</td>
<td>&gt;20</td>
<td></td>
</tr>
<tr>
<td>Total Peak Performance (CPU/GPU)</td>
<td>7.6/4.3</td>
<td>3/17</td>
<td></td>
</tr>
<tr>
<td>Number of CPU Chips</td>
<td>48,576</td>
<td>18,688</td>
<td></td>
</tr>
<tr>
<td>Number of GPU Chips</td>
<td>3,072</td>
<td>14,592</td>
<td></td>
</tr>
<tr>
<td>Amount of CPU Memory (TB)</td>
<td>1,510</td>
<td>688</td>
<td></td>
</tr>
<tr>
<td>Interconnect</td>
<td>3D Torus</td>
<td>3D Torus</td>
<td></td>
</tr>
<tr>
<td>Amount of On-line Disk Storage (PB)</td>
<td>26</td>
<td>20(?)</td>
<td></td>
</tr>
<tr>
<td>Sustained Disk Transfer (TB/sec)</td>
<td>&gt;1</td>
<td>0.4-0.7</td>
<td></td>
</tr>
<tr>
<td>Amount of Archival Storage</td>
<td>300</td>
<td>15-30</td>
<td></td>
</tr>
<tr>
<td>Sustained Tape Transfer (GB/sec)</td>
<td>100</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>
# Blue Waters and Kei Computing Systems

<table>
<thead>
<tr>
<th>System Attribute</th>
<th>NCSA Blue Waters</th>
<th>RIKEN Kei</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vendors</td>
<td>Cray/AMD/NVIDIA</td>
<td>Fujitsu</td>
</tr>
<tr>
<td>Processors</td>
<td>Interlagos/Kepler</td>
<td>SPARC64 VIIIifx</td>
</tr>
<tr>
<td>Total Peak Performance (PF)</td>
<td>11.9</td>
<td>11.3</td>
</tr>
<tr>
<td>Total Peak Performance (CPU/GPU)</td>
<td>7.6/4.3</td>
<td>11.3/0.0</td>
</tr>
<tr>
<td>Number of CPU Chips</td>
<td>48,576</td>
<td>88,128</td>
</tr>
<tr>
<td>Number of GPU Chips</td>
<td>3,072</td>
<td>0</td>
</tr>
<tr>
<td>Amount of CPU Memory (TB)</td>
<td>1,510</td>
<td>1,410</td>
</tr>
<tr>
<td>Interconnect</td>
<td>3D Torus</td>
<td>6D Torus</td>
</tr>
<tr>
<td>Amount of On-line Disk Storage (PB)</td>
<td>26</td>
<td>11/30</td>
</tr>
<tr>
<td>Sustained Disk Transfer (TB/sec)</td>
<td>&gt;1</td>
<td>?</td>
</tr>
<tr>
<td>Amount of Archival Storage</td>
<td>300</td>
<td>?</td>
</tr>
<tr>
<td>Sustained Tape Transfer (GB/sec)</td>
<td>100</td>
<td>?</td>
</tr>
</tbody>
</table>
## Blue Waters and Sequoia Computing Systems

<table>
<thead>
<tr>
<th>System Attribute</th>
<th>NCSA Blue Waters</th>
<th>LLNL Sequoia</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vendor(s)</td>
<td>Cray/AMD/NVIDIA</td>
<td>IBM</td>
</tr>
<tr>
<td>Processors</td>
<td>Interlagos/Kepler</td>
<td>PowerPCA2 variant</td>
</tr>
<tr>
<td>Total Peak Performance (PF)</td>
<td>11.9</td>
<td>20.1</td>
</tr>
<tr>
<td>Total Peak Performance (CPU/GPU)</td>
<td>7.6/4.3</td>
<td>20.1/0.0</td>
</tr>
<tr>
<td>Number of CPU Chips (8, 16 cores/chip)</td>
<td>48,576</td>
<td>98,304</td>
</tr>
<tr>
<td>Number of GPU Chips</td>
<td>3,072</td>
<td>0</td>
</tr>
<tr>
<td>Amount of CPU Memory (TB)</td>
<td>1,510</td>
<td>1,572</td>
</tr>
<tr>
<td>Interconnect</td>
<td>3D Torus</td>
<td>5D Torus</td>
</tr>
<tr>
<td>Amount of On-line Disk Storage (PB)</td>
<td>26</td>
<td>50(?)</td>
</tr>
<tr>
<td>Sustained Disk Transfer (TB/sec)</td>
<td>&gt;1</td>
<td>0.5-1.0</td>
</tr>
<tr>
<td>Amount of Archival Storage</td>
<td>300</td>
<td>?</td>
</tr>
<tr>
<td>Sustained Tape Transfer (GB/sec)</td>
<td>100</td>
<td>?</td>
</tr>
</tbody>
</table>
Petascale Computing Facility

- Modern Data Center
  - 90,000+ ft² total
  - 30,000 ft² raised floor
  - 20,000 ft² machine room gallery

- Energy Efficiency
  - LEED certified Gold
  - Power Utilization Efficiency
    - 1.1–1.2

Partners
- EYP MCF/
- Gensler
- IBM
- Yahoo!

19
Another Example System

- 128 node GPU Cluster
- #3 on Green500 in 2010
- Each node has
  - One Core i3 530 2.93 GHz dual-core CPU
  - One Tesla C2050 GPU per node
- 33.62 TFLOPS on HPL (10x ASCI Red)
- 934 MFLOPS/Watt
- But how do you program it?
An Even More Radical System

- Rack Scale
  - Processing: 128 Nodes, 1 (+) PF/s
  - Memory:
    - 128 TB DRAM
    - 0.4 PB/s Aggregate Bandwidth
  - NV Memory
    - 1 PB Phase Change Memory (addressable)
    - Additional 128 for Redundancy/RAID
- Network
  - 0.13 PB/sec Injection, 0.06 PB/s Bisection

<table>
<thead>
<tr>
<th>Deployment</th>
<th>Nodes</th>
<th>Topology</th>
<th>Compute</th>
<th>Mem BW</th>
<th>Injection BW</th>
<th>Bisection BW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module</td>
<td>1</td>
<td>N/A</td>
<td>8 TF/s</td>
<td>3 TB/s</td>
<td>1 TB/s</td>
<td>N/A</td>
</tr>
<tr>
<td>Deployable Cage</td>
<td>22</td>
<td>All-to-All</td>
<td>176 TF/s</td>
<td>67.5 TB/s</td>
<td>22.5 TB/s</td>
<td>31 TB/s</td>
</tr>
<tr>
<td>Rack</td>
<td>128</td>
<td>Flat. Butterfly</td>
<td>1 PF/s</td>
<td>.4 PB/s</td>
<td>0.13 PB/s</td>
<td>0.066 PB/s</td>
</tr>
<tr>
<td>Group Cluster</td>
<td>512</td>
<td>Flat. Butterfly</td>
<td>4.1 PF/s</td>
<td>1.6 PB/s</td>
<td>0.52 PB/s</td>
<td>0.26 PB/s</td>
</tr>
<tr>
<td>National Resource</td>
<td>128k</td>
<td>Hier. All-to-All</td>
<td>1 EF/s</td>
<td>0.4 EB/s</td>
<td>0.13 EB/s</td>
<td>16.8 PB/s</td>
</tr>
<tr>
<td>Max Configuration</td>
<td>2048k</td>
<td>Hier. All-to-All</td>
<td>16 EF/s</td>
<td>6.4 EB/s</td>
<td>2.1 EB/s</td>
<td>0.26 EB/s</td>
</tr>
</tbody>
</table>
How Do We Make Effective Use of These Systems?

- **Better use of our existing systems**
  - Blue Waters will provide a sustained PF, but that typically requires ~10PF peak

- **Improve node performance**
  - Make the compiler better
  - Give better code to the compiler
  - Get realistic with algorithms/data structures

- **Improve parallel performance/scalability**

- **Improve productivity of applications**
  - Better tools and interoperable languages, not a (single) new programming language

- **Improve algorithms**
  - Optimize for the real issues – data movement, power, resilience, ...
Make the Compiler Better

• It remains the case that most compilers cannot compete with hand-tuned or autotuned code on simple code
  ♦ Just look at dense matrix-matrix multiplication or matrix transpose
  ♦ Try it yourself!
    • Matrix multiply on my laptop:
    • N=100 (in cache): 1818 MF (1.1ms)
    • N=1000 (not): 335 MF (6s)
How Good are Compilers at Vectorizing Codes?

## Media Bench II Applications

<table>
<thead>
<tr>
<th>Appl</th>
<th>XLC</th>
<th>ICC</th>
<th>GCC</th>
<th>XLC</th>
<th>ICC</th>
<th>GCC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JPEG Enc</td>
<td>-</td>
<td>1.33</td>
<td>-</td>
<td>1.39</td>
<td>2.13</td>
<td>1.57</td>
</tr>
<tr>
<td>JEPG Dec</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.14</td>
<td>1.13</td>
</tr>
<tr>
<td>H263 Enc</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.25</td>
<td>2.28</td>
<td>2.06</td>
</tr>
<tr>
<td>H263 Dec</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.31</td>
<td>1.45</td>
<td>-</td>
</tr>
<tr>
<td>MPEG2 Enc</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.06</td>
<td>1.96</td>
<td>2.43</td>
</tr>
<tr>
<td>MPEG2 Dec</td>
<td>-</td>
<td>-</td>
<td>1.15</td>
<td>1.37</td>
<td>1.45</td>
<td>1.55</td>
</tr>
<tr>
<td>MPEG4 Enc</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.44</td>
<td>1.81</td>
<td>1.74</td>
</tr>
<tr>
<td>MPEG4 Dec</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.12</td>
<td>-</td>
<td>1.18</td>
</tr>
</tbody>
</table>

Table shows **whole program speedups** measured against unvectorized application

How Do We Change This?

- Test compiler against “equivalent” code (e.g., best hand-tuned or autotuned code that performs the same computation, under some interpretation or “same”)
  - In a perfect world, the compiler would provide the same, excellent performance for all equivalent versions
- As part of the Blue Waters project, Padua, Garzaran, Maleki are developing a test suite that evaluates how the compiler does with such equivalent code
  - Working with vendors to improve the compiler
  - Identify necessary transformations
  - Identify opportunities for better interaction with the programmer to facilitate manual intervention.
  - Main focus has been on code generation for vector extensions
  - Result is a compiler whose realized performance is less sensitive to different expression of code and therefore closer to that of the best hand-tuned code.
  - Just by improving automatic vectorization, loop speedups of more than 5 have been observed on the Power 7.
- But this is a long-term project
  - What can we do in the meantime?
Give “Better” Code to the Compiler

- Augmenting current programming models and languages to exploit advanced techniques for performance optimization (i.e., autotuning)
- Not a new idea, and some tools already do this.
- But how can these approaches become part of the mainstream development?
How Can Autotuning Tools Fit Into Application Development?

• In the short run, just need effective mechanisms to replace user code with tuned code
  ♦ Manual extraction of code, specification of specific collections of code transformations

• But this produces at least two versions of the code (tuned (for a particular architecture and problem choice) and untuned). And there are other issues.

• What does an application want (what is the Dream)?
Application Needs Include

- Code must be portable
- Code must be persistent
- Code must permit (and encourage) experimentation
- Code must be maintainable
- Code must be correct
- Code must be faster
Implications of These Requirements

- **Portable** - augment existing language. Either use pragmas/comments or extremely portable precompiler
  - Best if the tool that performs all of these steps looks like just like the compiler, for integration with build process

- **Persistent**
  - Keep original and transformed code around: *Golden Copy*

- **Maintainable**
  - Let user work with original code *and* ensure changes automatically update tuned code

- **Correct**
  - Do whatever the application developer needs to believe that the tuned code is correct
    - In the end, this **will** require running some comparison tests

- **Faster**
  - Must be able to interchange tuning tools - pick the best tool for *each* part of the code
  - No captive interfaces
  - Extensibility - a clean way to add new tools, transformations, properties, ...
Application-Relevant Abstractions

• Language for interfacing with autotuning must convey concepts that are meaningful to the application programmer

• Wrong: unroll by 5
  ♦ Though could be ok for performance expert, and some compilers already provide pragmas for specific transformations

• Right (maybe): Performance precious, typical loop count between 100 and 10000, even, not power of 2

• We need work at developing higher-level, performance-oriented languages or language extensions
  ♦ This would be the “good” future
Better Algorithms and Data Structures

- Autotuning only offers the best performance with the given data structure and algorithm
  - That’s a big constraint
- Processors include hardware to address performance challenges
  - “Vector” function units
  - Memory latency hiding/prefetch
  - Atomic update features for shared memory
  - Etc.
Sparse Matrix-Vector Multiply

Barriers to faster code

- “Standard” formats such as CSR do not meet requirements for prefetch or vectorization
- Modest changes to data structure enable both vectorization, prefetch, for 20-80% improvement on P7

Prefetch results in Optimizing Sparse Data Structures for Matrix Vector Multiply http://hpc.sagepub.com/content/25/1/115
What Does This Mean For You?

- It is time to rethink data structures and algorithms to match the realities of memory architecture
  - We have results for x86 where the benefit is smaller but still significant
  - Better match of algorithms to prefetch hardware is necessary to overcome memory performance barriers
- Similar issues come up with heterogeneous processing elements (someone needs to design for memory motion and concurrent and nonblocking data motion)
Performance on a Node

• Nodes are SMPs
  ♦ You have this problem on anything (even laptops)

• Tuning issues include the usual
  ♦ Getting good performance out of the compiler (often means adapting to the memory hierarchy)

• New (SMP) issues include
  ♦ Sharing the SMP with other processes
  ♦ Sharing the memory system
New (?) Wrinkle – Avoiding Jitter

• Jitter here means the variation in time measured when running identical computations
  ♦ Caused by other computations, e.g., an OS interrupt to handle a network event or runtime library servicing a communication or I/O request

• This problem is in some ways less serious on HPC platform, as the OS and runtime services are tuned to minimize impact
  ♦ However, cannot be eliminated entirely
Sharing an SMP

- Having many cores available makes everyone think that they can use them to solve other problems (“no one would use all of them all of the time”)
- However, compute-bound scientific calculations are often written as if all compute resources are owned by the application
- Such static scheduling leads to performance loss
- Pure dynamic scheduling adds overhead, but is better
- Careful mixed strategies are even better
- Thanks to Vivek Kale
Happy Medium Scheduling

Communication Avoiding LU factorization (CALU) algorithm

S. Donfack, L. Grigori, V. Kale, WG, IPDPS '12
Synchronization and OS Noise

• “Characterizing the Influence of System Noise on Large-Scale Applications by Simulation,” Torsten Hoefler, Timo Schneider, Andrew Lumsdaine
  ♦ Best Paper, SC10
• Next 3 slides based on this talk...
A Noisy Example – Dissemination Barrier

- Process 4 is delayed
  - Noise propagates “wildly” (of course deterministic)
Single Collective Operations and Noise

- 1 Byte, Dissemination, regular noise, 1000 Hz, 100 µs
The problem is *blocking* operations

- Simple, data-parallel algorithms easy to reason about but inefficient
  - True for decades, but ignored (memory)
- One solution: fully asynchronous methods
  - Very attractive, yet efficiency is low and there are good reasons for that
  - Blocking can be due to fully collective (e.g., Allreduce) or neighbor communications (halo exchange)
  - Can we save methods that involve global, synchronizing operations?
Saving Allreduce

• One common suggestion is to avoid using Allreduce
  ♦ But algorithms with dot products are among the best known
  ♦ Can sometimes aggregate the data to reduce the number of separate Allreduce operations
  ♦ But better is to reduce the impact of the synchronization by hiding the Allreduce behind other operations (in MPI, using MPI_Iallreduce)

• We can adapt CG to nonblocking Allreduce with some added floating point (but perhaps little time cost)
The Conjugate Gradient Algorithm

• While (not converged)
  niter += 1;
  s = A * p;
  t = p' * s;
  alpha = gmma / t;
  x = x + alpha * p;
  r = r - alpha * s;
  if rnorm2 < tol2 ; break ; end
  z = M * r;
  gmmaNew = r' * z;
  beta = gmmaNew / gmma;
  gmma = gmmaNew;
  p = z + beta * p;
end
The Conjugate Gradient Algorithm

- While (not converged)
  
  nIter += 1;
  
  s = A * p;
  
  t = p' * s;
  
  alpha = gamma / t;
  
  x = x + alpha * p;
  
  r = r - alpha * s;
  
  if rnorm2 < tol2 ; break ; end
  
  z = M * r;
  
  gammaNew = r' * z;
  
  beta = gammaNew / gamma;
  
  gamma = gammaNew;
  
  p = z + beta * p;
  
  end
CG Reconsidered

- By reordering operations, nonblocking dot products (MPI_Iallreduce in MPI-3) can be overlapped with other operations.
- Trades extra local work for overlapped communication:
  - On a pure floating point basis, the nonblocking version requires 2 more DAXPY operations.
  - A closer analysis shows that some operations can be merged.
- More work does not imply more time.
What’s Different at Peta/Exascale

• Performance Focus
  ♦ Only a little – basically, the resource is expensive, so a premium placed on making good use of resource
  ♦ Quite a bit – node is more complex, has more features that must be exploited

• Scalability
  ♦ Solutions that work at 100-1000 way often inefficient at 100,000-way
  ♦ Some algorithms scale well
    • Explicit time marching in 3D
  ♦ Some don’t
    • Direct implicit methods
  ♦ Some scale well for a while
    • FFTs (communication volume in Alltoall)
  ♦ Load balance, latency are critical issues

• Fault Tolerance becoming important
  ♦ Now: Reduce time spent in checkpoints
  ♦ Soon: Lightweight recovery from transient errors
Preparing for the Next Generation of HPC Systems

- Better use of existing resources
  - Performance-oriented programming
  - Dynamic management of resources at all levels
  - Embrace hybrid programming models (you have already if you use SSE/VSX/OpenMP/...)

- Focus on results
  - Adapt to available network bandwidth and latency
  - Exploit I/O capability (available space crew faster than processor performance!)

- Prepare for the future
  - Fault tolerance
  - Hybrid processor architectures
  - Latency tolerant algorithms
  - Data-driven systems
Recommended Reading

- Bit reversal on uniprocessors (Alan Karp, SIAM Review, 1996)
- Experimental Analysis of Algorithms (Catherine McGeoch, Notices of the American Mathematical Society, March 2001)
- Reflections on the Memory Wall (Sally McKee, ACM Conference on Computing Frontiers, 2004)
Thanks

- Torsten Hoefler
  - Performance modeling lead, Blue Waters; MPI datatype
- David Padua, Maria Garzaran, Saeed Maleki
  - Compiler vectorization
- Dahai Guo
  - Streamed format exploiting prefetch, vectorization, GPU
- Vivek Kale
  - SMP work partitioning
- Hormozd Gahvari
  - AMG application modeling
- Marc Snir and William Kramer
  - Performance model advocates
- Abhinav Bhatele
  - Process/node mapping
- Elena Caraba
  - Nonblocking Allreduce in CG
- Van Bui
  - Performance model-based evaluation of programming models
- Funding provided by:
  - Blue Waters project (State of Illinois and the University of Illinois)
  - Department of Energy, Office of Science
  - National Science Foundation