Impact of Architecture and Technology for Extreme Scale on Software and Algorithm Design

Jack Dongarra

University of Tennessee
Oak Ridge National Laboratory
University of Manchester

9/8/2010
- Listing of the 500 most powerful Computers in the World
- Yardstick: Rmax from LINPACK MPP

Ax = b, dense problem

- Updated twice a year
  SC‘xy in the States in November
  Meeting in Germany in June

- All data available from www.top500.org
Performance Development

- 1.76 PFlop/s
- 32.4 PFlop/s
- 24.7 TFlop/s
- 3.24 PFlop/s
- 59.7 GFlop/s
- 1.17 TFlop/s
- N=1
- N=500
- 400 MFlop/s
- My Laptop
- My iPhone (40 MFlop/s)

SUM

- 10000
- 100000
- 1000000
- 10000000
- 1 PFlop/s
- 100 Pflop/s
- 100000000000 MFlop/s

- 1993 1995 1997 1999 2001 2003 2005 2007 2009

- 6-8 years

- N=1

- N=500
Processors Used in the Top500 Systems

- Intel 81%
- AMD 10%
- IBM 8%

[Pie chart showing processor usage: Intel EM64T, Others, Intel IA-64, Power, AMD x86_64]
Today’s Multicores

99% of Top500 Systems Are Based on Multicore

Of the Top500, 499 are multicore.
Performance of Countries

Total Performance [Tflop/s]

US

Performance of Countries

Total Performance [Tflop/s]

US
EU
Japan
China

Countries / System Share

7 systems in the Italy
## June 2010: The TOP10

<table>
<thead>
<tr>
<th>Rank</th>
<th>Site</th>
<th>Computer</th>
<th>Country</th>
<th>Cores</th>
<th>Rmax [Pflops]</th>
<th>% of Peak</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DOE / OS Oak Ridge Nat Lab</td>
<td>Jaguar / Cray Cray XT5sixCore 2.6 GHz</td>
<td>USA</td>
<td>224,162</td>
<td>1.76</td>
<td>75</td>
</tr>
<tr>
<td>2</td>
<td>Nat. Supercomputer Center in Shenzhen</td>
<td>Nebulea / Dawning / TC3600 Blade, Intel X5650, Nvidia C2050 GPU</td>
<td>China</td>
<td>120,640</td>
<td>1.27</td>
<td>43</td>
</tr>
<tr>
<td>3</td>
<td>DOE / NNSA Los Alamos Nat Lab</td>
<td>Roadrunner / IBM BladeCenterQS22/LS21</td>
<td>USA</td>
<td>122,400</td>
<td>1.04</td>
<td>76</td>
</tr>
<tr>
<td>4</td>
<td>NSF / NICS / U of Tennessee</td>
<td>Kraken/ Cray Cray XT5sixCore 2.6 GHz</td>
<td>USA</td>
<td>98,928</td>
<td>.831</td>
<td>81</td>
</tr>
<tr>
<td>5</td>
<td>ForschungszentrumJülich (FZJ)</td>
<td>Jugene / IBM Blue Gene/P Solution</td>
<td>Germany</td>
<td>294,912</td>
<td>.825</td>
<td>82</td>
</tr>
<tr>
<td>6</td>
<td>NASA / Ames Research Center/NAS</td>
<td>Pleiades / SGI SGI Altix ICE 8200EX</td>
<td>USA</td>
<td>56,320</td>
<td>.544</td>
<td>82</td>
</tr>
<tr>
<td>7</td>
<td>National SC Center in Tianjin / NUDT</td>
<td>Tianhe-1 / NUDT TH-1 / IntelQC + AMD ATI Radeon 4870</td>
<td>China</td>
<td>71,680</td>
<td>.563</td>
<td>46</td>
</tr>
<tr>
<td>8</td>
<td>DOE / NNSA Lawrence Livermore NL</td>
<td>BlueGene/L IBM eServerBlue Gene Solution</td>
<td>USA</td>
<td>212,992</td>
<td>.478</td>
<td>80</td>
</tr>
<tr>
<td>9</td>
<td>DOE / OS Argonne Nat Lab</td>
<td>Intrepid / IBM Blue Gene/P Solution</td>
<td>USA</td>
<td>163,840</td>
<td>.458</td>
<td>82</td>
</tr>
<tr>
<td>10</td>
<td>DOE / NNSA Sandia Nat Lab</td>
<td>Red Sky / Sun / SunBlade 6275</td>
<td>USA</td>
<td>42,440</td>
<td>.433</td>
<td>87</td>
</tr>
</tbody>
</table>
## June 2010: The TOP10

<table>
<thead>
<tr>
<th>Rank</th>
<th>Site</th>
<th>Computer</th>
<th>Country</th>
<th>Cores</th>
<th>Rmax [Pflops]</th>
<th>% of Peak</th>
<th>Power [MW]</th>
<th>MFlops/Watt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DOE / OS Oak Ridge Nat Lab</td>
<td>Jaguar / Cray XT5sixCore 2.6 GHz</td>
<td>USA</td>
<td>224,162</td>
<td>1.76</td>
<td>75</td>
<td>7.0</td>
<td>251</td>
</tr>
<tr>
<td>2</td>
<td>Nat. Supercomputer Center in Shenzhen</td>
<td>Nebulea / Dawning / TC3600 Blade, Intel X5650, Nvidia C2050 GPU</td>
<td>China</td>
<td>120,640</td>
<td>1.27</td>
<td>43</td>
<td>2.58</td>
<td>493</td>
</tr>
<tr>
<td>3</td>
<td>DOE / NNSA Los Alamos Nat Lab</td>
<td>Roadrunner / IBM BladeCenterQS22/LS21</td>
<td>USA</td>
<td>122,400</td>
<td>1.04</td>
<td>76</td>
<td>2.48</td>
<td>446</td>
</tr>
<tr>
<td>4</td>
<td>NSF / NICS / U of Tennessee</td>
<td>Kraken/ Cray XT5sixCore 2.6 GHz</td>
<td>USA</td>
<td>98,928</td>
<td>.831</td>
<td>81</td>
<td>3.09</td>
<td>269</td>
</tr>
<tr>
<td>5</td>
<td>ForschungszentrumJulich (FZJ)</td>
<td>Jugene / IBM Blue Gene/P Solution</td>
<td>Germany</td>
<td>294,912</td>
<td>.825</td>
<td>82</td>
<td>2.26</td>
<td>365</td>
</tr>
<tr>
<td>6</td>
<td>NASA / Ames Research Center/NAS</td>
<td>Pleiades / SGI Altix ICE 8200EX</td>
<td>USA</td>
<td>56,320</td>
<td>.544</td>
<td>82</td>
<td>3.1</td>
<td>175</td>
</tr>
<tr>
<td>7</td>
<td>National SC Center in Tianjin / NUDT</td>
<td>Tianhe-1 / NUDT TH-1 / IntelQC + AMD ATI Radeon 4870</td>
<td>China</td>
<td>71,680</td>
<td>.563</td>
<td>46</td>
<td>1.48</td>
<td>380</td>
</tr>
<tr>
<td>8</td>
<td>DOE / NNSA Lawrence Livermore NL</td>
<td>BlueGene/L IBM eServerBlue Gene Solution</td>
<td>USA</td>
<td>212,992</td>
<td>.478</td>
<td>80</td>
<td>2.32</td>
<td>206</td>
</tr>
<tr>
<td>9</td>
<td>DOE / OS Argonne Nat Lab</td>
<td>Intrepid / IBM Blue Gene/P Solution</td>
<td>USA</td>
<td>163,840</td>
<td>.458</td>
<td>82</td>
<td>1.26</td>
<td>363</td>
</tr>
<tr>
<td>10</td>
<td>DOE / NNSA Sandia Nat Lab</td>
<td>Red Sky / Sun / SunBlade 6275</td>
<td>USA</td>
<td>42,440</td>
<td>.433</td>
<td>87</td>
<td>2.4</td>
<td>180</td>
</tr>
</tbody>
</table>
#1 ORNL’s Newest System Jaguar XT5

Recently upgraded to a 2 Pflop/s system with more than 224K cores using AMD’s 6 Core chip.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peak performance</td>
<td>2.332 PF</td>
</tr>
<tr>
<td>System memory</td>
<td>300 TB</td>
</tr>
<tr>
<td>Disk space</td>
<td>10 PB</td>
</tr>
<tr>
<td>Disk bandwidth</td>
<td>240+ GB/s</td>
</tr>
<tr>
<td>Interconnect bandwidth</td>
<td>374 TB/s</td>
</tr>
</tbody>
</table>
Nebulae

Hybrid system, commodity + GPUs

Theoretical peak 2.98 Pflop/s

Linpack Benchmark at 1.27 Pflop/s

4640 nodes, each node:
2 Intel 6-core Xeon5650 + Nvidia Fermi C2050 GPU (each 14 cores)

- 120,640 cores
- Infiniband connected

- 500 MB/s peak per link and 8 GB/s
Commodity plus Accelerators

**Commodity**
- Intel Xeon
- 8 cores
- 3 GHz
- 8*4 ops/cycle
- 96 Gflop/s (DP)

**Accelerator (GPU)**
- Nvidia C2050 “Fermi”
- 448 “Cuda cores”
- 1.15 GHz
- 448 ops/cycle
- 515 Gflop/s (DP)

---

Interconnect PCI Express
- 512 MB/s to 32GB/s
- 8 MW – 512 MW
Looking at the Gordon Bell Prize
(Recognize outstanding achievement in high-performance computing applications and encourage development of parallel processing)

- 1 GFlop/s; 1988; Cray Y-MP; 8 Processors
  - Static finite element analysis
- 1 TFlop/s; 1998; Cray T3E; 1024 Processors
  - Modeling of metallic magnet atoms, using a variation of the locally self-consistent multiple scattering method.
- 1 PFlop/s; 2008; Cray XT5; 1.5x10^5 Processors
  - Superconductive materials
- 1 EFlop/s; ~2018; ?; 1x10^7 Processors (10^9 threads)
Performance Development in Top500

SUM

Gordon Bell Winners

1 Gflop/s 100 Mflop/s 10 Gflop/s 100 Gflop/s 100 Tflop/s 10 Pflop/s 1 Tflop/s 1 Pflop/s 1 Eflop/s

### Potential System Architecture

with a cap of $200M and 20MW

<table>
<thead>
<tr>
<th>Systems</th>
<th>2010</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System peak</strong></td>
<td>2 Pflop/s</td>
</tr>
<tr>
<td><strong>Power</strong></td>
<td>6 MW</td>
</tr>
<tr>
<td><strong>System memory</strong></td>
<td>0.3 PB</td>
</tr>
<tr>
<td><strong>Node performance</strong></td>
<td>125 GF</td>
</tr>
<tr>
<td><strong>Node memory BW</strong></td>
<td>25 GB/s</td>
</tr>
<tr>
<td><strong>Node concurrency</strong></td>
<td>12</td>
</tr>
<tr>
<td><strong>Total Node Interconnect BW</strong></td>
<td>3.5 GB/s</td>
</tr>
<tr>
<td><strong>System size (nodes)</strong></td>
<td>18,700</td>
</tr>
<tr>
<td><strong>Total concurrency</strong></td>
<td>225,000</td>
</tr>
<tr>
<td><strong>Storage</strong></td>
<td>15 PB</td>
</tr>
<tr>
<td><strong>IO</strong></td>
<td>0.2 TB</td>
</tr>
<tr>
<td><strong>MTTI</strong></td>
<td>days</td>
</tr>
</tbody>
</table>
## Potential System Architecture with a cap of $200M and 20MW

<table>
<thead>
<tr>
<th>Systems</th>
<th>2010</th>
<th>2018</th>
</tr>
</thead>
<tbody>
<tr>
<td>System peak</td>
<td>2 Pflop/s</td>
<td>1 Eflop/s</td>
</tr>
<tr>
<td><strong>Power</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power</td>
<td>6 MW</td>
<td>~20 MW</td>
</tr>
<tr>
<td>System memory</td>
<td>0.3 PB</td>
<td>32 - 64 PB [0.03 Bytes/Flop]</td>
</tr>
<tr>
<td>Node performance</td>
<td>125 GF</td>
<td>1,2 or 15TF</td>
</tr>
<tr>
<td>Node memory BW</td>
<td>25 GB/s</td>
<td>2 - 4TB/s [0.002 Bytes/Flop]</td>
</tr>
<tr>
<td>Node concurrency</td>
<td>12</td>
<td>O(1k) or 10k</td>
</tr>
<tr>
<td>Total Node Interconnect BW</td>
<td>3.5 GB/s</td>
<td>200-400GB/s (1:4 or 1:8 from memory BW)</td>
</tr>
<tr>
<td>System size (nodes)</td>
<td>18,700</td>
<td>O(100,000) or O(1M)</td>
</tr>
<tr>
<td>Total concurrency</td>
<td>225,000</td>
<td>O(billion) [O(10) to O(100) for latency hiding]</td>
</tr>
<tr>
<td>Storage</td>
<td>15 PB</td>
<td>500-1000 PB (&gt;10x system memory is min)</td>
</tr>
<tr>
<td>IO</td>
<td>0.2 TB</td>
<td>60 TB/s (how long to drain the machine)</td>
</tr>
<tr>
<td>MTTI</td>
<td>days</td>
<td>O(1 day)</td>
</tr>
</tbody>
</table>
# Potential System Architecture with a cap of $200M and 20MW

<table>
<thead>
<tr>
<th>Systems</th>
<th>2010</th>
<th>2018</th>
<th>Difference Today &amp; 2018</th>
</tr>
</thead>
<tbody>
<tr>
<td>System peak</td>
<td>2 Pflop/s</td>
<td>1 Eflop/s</td>
<td>O(1000)</td>
</tr>
<tr>
<td>Power</td>
<td>6 MW</td>
<td>~20 MW</td>
<td></td>
</tr>
<tr>
<td>System memory</td>
<td>0.3 PB</td>
<td>32 - 64 PB [ .03 Bytes/Flop ]</td>
<td>O(100)</td>
</tr>
<tr>
<td>Node performance</td>
<td>125 GF</td>
<td>1,2 or 15TF</td>
<td>O(10) - O(100)</td>
</tr>
<tr>
<td>Node memory BW</td>
<td>25 GB/s</td>
<td>2 - 4TB/s [ .002 Bytes/Flop ]</td>
<td>O(100)</td>
</tr>
<tr>
<td>Node concurrency</td>
<td>12</td>
<td>O(1k) or 10k</td>
<td>O(100) - O(1000)</td>
</tr>
<tr>
<td>Total Node Interconnect BW</td>
<td>3.5 GB/s</td>
<td>200-400GB/s (1:4 or 1:8 from memory BW)</td>
<td>O(100)</td>
</tr>
<tr>
<td>System size (nodes)</td>
<td>18,700</td>
<td>O(100,000) or O(1M)</td>
<td>O(10) - O(100)</td>
</tr>
<tr>
<td>Total concurrency</td>
<td>225,000</td>
<td>O(billion) [O(10) to O(100) for latency hiding]</td>
<td>O(10,000)</td>
</tr>
<tr>
<td>Storage</td>
<td>15 PB</td>
<td>500-1000 PB (&gt;10x system memory is min)</td>
<td>O(10) - O(100)</td>
</tr>
<tr>
<td>IO</td>
<td>0.2 TB</td>
<td>60 TB/s (how long to drain the machine)</td>
<td>O(100)</td>
</tr>
<tr>
<td>MTTI</td>
<td>days</td>
<td>O(1 day)</td>
<td>- O(10)</td>
</tr>
</tbody>
</table>
Exascale ($10^{18}$ Flop/s) Systems: Two possible paths

- **Light weight processors (think BG/P)**
  - ~1 GHz processor ($10^9$)
  - ~1 Kilo cores/socket ($10^3$)
  - ~1 Mega sockets/system ($10^6$)

- **Hybrid system (think GPU based)**
  - ~1 GHz processor ($10^9$)
  - ~10 Kilo FPUs/socket ($10^4$)
  - ~100 Kilo sockets/system ($10^5$)
Factors that Necessitate Redesign of Our Software

- Steepness of the ascent from terascale to petascale to exascale
- Extreme parallelism and hybrid design
  - Preparing for million/billion way parallelism
- Tightening memory/bandwidth bottleneck
  - Limits on power/clock speed implication on multicore
  - Reducing communication will become much more intense
  - Memory per core changes, byte-to-flop ratio will change
- Necessary Fault Tolerance
  - MTTF will drop
  - Checkpoint/restart has limitations

Average Number of Cores Per Supercomputer for Top20 Systems

Software infrastructure does not exist today
Moore’s Law reinterpreted

- Number of cores per chip will double every two years
- Clock speed will not increase (possibly decrease) because of Power

- Need to deal with systems with millions of concurrent threads
- Need to deal with inter-chip parallelism as well as intra-chip parallelism
Major Changes to Software

- **Must rethink the design of our software**
  - Another disruptive technology
    - Similar to what happened with cluster computing and message passing
  - Rethink and rewrite the applications, algorithms, and software

- **Numerical libraries for example will change**
  - For example, both LAPACK and ScaLAPACK will undergo major changes to accommodate this
Future Computer Systems

- Most likely be a hybrid design
- Think standard multicore chips and accelerator (GPUs)
- Today accelerators are attached
- Next generation more integrated
- Intel’s Larrabee? Now called “Knights Corner” and “Knights Ferry” to come.
  - 48 x86 cores
- AMD’s Fusion in 2011 - 2013
  - Multicore with embedded graphics ATI
- Nvidia’s plans?
What’s Next?

Different Classes of Chips
- Home
- Games / Graphics
- Business
- Scientific
Five Important Software Features to Consider When Computing at Scale

1. Effective Use of Many-Core and Hybrid architectures
   - Break fork-join parallelism
   - Dynamic Data Driven Execution
   - Block Data Layout

2. Exploiting Mixed Precision in the Algorithms
   - Single Precision is 2X faster than Double Precision
   - With GP-GPUs 10x
   - Power saving issues

3. Self Adapting / Auto Tuning of Software
   - Too hard to do by hand

4. Fault Tolerant Algorithms
   - With 1,000,000’s of cores things will fail

5. Communication Reducing Algorithms
   - For dense computations from $O(n \log p)$ to $O(\log p)$ communications
   - Asynchronous iterations
   - GMRES k-step compute ($x, Ax, A^2x, \ldots A^kx$)
• Fork-join, bulk synchronous processing
Parallel Tasks in $LU/LL^T/QR$

- Break into smaller tasks and remove dependencies

* LU does block pair wise pivoting
PLASMA: Parallel Linear Algebra s/w for Multicore Architectures

• Objectives
  - High utilization of each core
  - Scaling to large number of cores
  - Shared or distributed memory

• Methodology
  - Dynamic DAG scheduling
  - Explicit parallelism
  - Implicit communication
  - Fine granularity / block data layout

• Arbitrary DAG with dynamic scheduling

![DAG Diagram]

Cholesky 4 x 4

Fork-join parallelism

DAG scheduled parallelism

Time
Communication Avoiding Algorithms

- **Goal:** Algorithms that communicate as little as possible
- Jim Demmel and company have been working on algorithms that obtain a provable minimum communication.
- **Direct methods (BLAS, LU, QR, SVD, other decompositions)**
  - Communication lower bounds for *all* these problems
  - Algorithms that attain them (*all* dense linear algebra, some sparse)
    - Mostly not in LAPACK or ScaLAPACK (yet)
- **Iterative methods - Krylov subspace methods for** $Ax=b$, $Ax=\lambda x$
  - Communication lower bounds, and algorithms that attain them (depending on sparsity structure)
    - Not in any libraries (yet)
- **For QR Factorization they can show:**

<table>
<thead>
<tr>
<th># flops</th>
<th>Lower bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Theta(mn^2)$</td>
<td>$\Theta(mn^2)$</td>
</tr>
<tr>
<td>$\Theta(n \sqrt{W})$</td>
<td>$\Theta(n \sqrt{W})$</td>
</tr>
<tr>
<td>$\Theta(mn^2/W)$</td>
<td>$\Theta(mn^2/W^{3/2})$</td>
</tr>
</tbody>
</table>
Standard QR Block Reduction

- We have a $mxn$ matrix $A$ we want to reduce to upper triangular form.
Standard QR Block Reduction

- We have a \( mxn \) matrix \( A \) we want to reduce to upper triangular form.

\[ Q_1^T \longrightarrow 1 \]
Standard QR Block Reduction

- We have a $mxn$ matrix $A$ we want to reduce to upper triangular form.

$$Q_1^T \rightarrow Q_2^T$$
Standard QR Block Reduction

- We have a $mxn$ matrix $A$ we want to reduce to upper triangular form.

$Q_1^T \rightarrow Q_2^T \rightarrow \text{a triangular matrix}$
• We have a $mxn$ matrix $A$ we want to reduce to upper triangular form.
We have a $mxn$ matrix $A$ we want to reduce to upper triangular form.
We have a $mxn$ matrix $A$ we want to reduce to upper triangular form.

$$A = Q_1 Q_2 Q_3 R = QR$$
Communication Avoiding QR Example

Communication Avoiding QR Example

Communication Reducing QR Factorization

Theoretical peak is 153.2 Gflop/s with 16 cores. Matrix size 51200 by 3200.
Challenges of using GPUs

- **High levels of parallelism**
  Many GPU cores, serial kernel execution
  [ e.g. 240 in the Nvidia Tesla; up to 512 in Fermi - to have concurrent kernel execution ]

- **Hybrid/heterogeneous architectures**
  Match algorithmic requirements to architectural strengths
  [ e.g. small, non-parallelizable tasks to run on CPU, large and parallelizable on GPU ]

- **Compute vs communication gap**
  Exponentially growing gap; persistent challenge
  [ Processor speed improves 59%, memory bandwidth 23%, latency 5.5% ]
  [ on all levels, e.g. a GPU Tesla C1070 (4 x C1060) has compute power of O(1,000) Gflop/s but GPUs communicate through the CPU using O(1) GB/s connection ]
Hybrid Computing

- Match algorithmic requirements to architectural strengths of the hybrid components
  - Multicore: small tasks/tiles
  - Accelerator: large data parallel tasks

  e.g. split the computation into tasks; define critical path that “clears” the way for other large data parallel tasks; proper schedule the tasks execution

  Design algorithms with well defined “search space” to facilitate auto-tuning
Cholesky on **multicore + multi-GPUs**

**Hardware**

- **HOST**: Two-dual core AMD Opteron 1.8GHz, **2GB** memory
- **DEVICE**:
  - 4 GPU TESLA C1070 1.44GHz
  - 240 computing cores per GPU
  - 4GB memory per GPU
  - Single precision floating point performance (NVIDIA PEAK): 4.14 Tflop/s
  - Memory bandwidth: 408 GB/s
  - System interface: PCIeexpress

**Memory limitations prevented runs on larger sizes**
SP Cholesky on Multicore + Multi GPUs

Parallel Performance of the hybrid SPOTRF (4 Opteron 1.8GHz and 4 GPU TESLA C1060 1.44GHz)

Matrix sizes vs. Gflop/s for different configurations of CPUs and GPUs:
- 1CPU-1GPU
- 2CPUs-2GPUs
- 3CPUs-3GPUs
- 4CPUs-4GPUs

The graph shows how the performance (measured in Gflop/s) changes with increasing matrix sizes for each configuration.
Performance of Single Precision on Conventional and GPU’s

- Realized have the similar situation on our commodity processors.
  - That is, SP is 2X as fast as DP on many systems

- The Intel Xeon and AMD Opteron have SSE3
  - 2 flops/cycle DP
  - 4 flops/cycle SP

- IBM PowerPC has AltiVec
  - 8 flops/cycle SP
  - 4 flops/cycle DP
    - No DP on AltiVec

NVIDIA Tesla

Best case reality: 240 mul-adds per clock
  Just able to do the mul-add so 2/3 or 624 Gflop/s of theoretical peak

All this is single precision
  Double precision is 78 Gflop/s peak
  (Factor of 8 from SP; exploit mixed prec)

Single precision is faster because:
- Operations are faster
- Reduced data motion
- Larger blocks gives higher locality in cache
Idea Goes Something Like This...

- Exploit 32 bit floating point as much as possible.
  - Especially for the bulk of the computation
- Correct or update the solution with selective use of 64 bit floating point to provide a refined results
- Intuitively:
  - Compute a 32 bit result,
  - Calculate a correction to 32 bit result using selected higher precision and,
  - Perform the update of the 32 bit results with the correction using high precision.
Mixed-Precision Iterative Refinement

- Iterative refinement for dense systems, $Ax = b$, can work this way.

\[
L \ U = lu(A)\text{SINGLE } O(n^3)
\]
\[
x = L\backslash(U\backslash b)\text{SINGLE } O(n^2)
\]
\[
r = b - Ax\text{DOUBLE } O(n^2)
\]

WHILE $||r||$ not small enough

\[
z = L\backslash(U\backslash r)
\]
\[
x = x + z\text{DOUBLE } O(n^1)
\]
\[
r = b - Ax\text{DOUBLE } O(n^2)
\]

END

- Wilkinson, Moler, Stewart, & Higham provide error bound for SP fl pt results when using DP fl pt.
Mixed-Precision Iterative Refinement

Iterative refinement for dense systems, $Ax = b$, can work this way.

$L U = lu(A)$ \text{SINGLE} $O(n^3)$
$x = L\backslash(U\backslash b)$ \text{SINGLE} $O(n^2)$
$r = b - Ax$ \text{DOUBLE} $O(n^2)$

\textbf{WHILE} $||r||$ not small enough

$z = L\backslash(U\backslash r)$ \text{\textbf{SINGLE}} $O(n^2)$
$x = x + z$ \text{\textbf{DOUBLE}} $O(n^1)$
$r = b - Ax$ \text{\textbf{DOUBLE}} $O(n^2)$
END

- Wilkinson, Moler, Stewart, & Higham provide error bound for SP fl pt results when using DP fl pt.
- It can be shown that using this approach we can compute the solution to 64-bit floating point precision.

- Requires extra storage, total is 1.5 times normal;
- $O(n^3)$ work is done in lower precision
- $O(n^2)$ work is done in high precision
- Problems if the matrix is ill-conditioned in sp; $O(10^8)$
Results for Mixed Precision Iterative Refinement for Dense Ax = b

- Single precision is faster than DP because:
  - Higher parallelism within floating point units
    - 4 ops/cycle (usually) instead of 2 ops/cycle
  - Reduced data motion
    - 32 bit data instead of 64 bit data
  - Higher locality in cache
    - More data items in cache
\[ Ax = b \]

**Single Precision**

**Double Precision**

Tesla C2050, 448 CUDA cores (14 multiprocessors \(x\) 32) \(\text{at} 1.15\text{GHz.}\),

3 GB memory, connected through PCIe to a quad-core Intel \(\text{at} 2.5\text{GHz.}\).
Ax = b

Tesla C2050, 448 CUDA cores (14 multiprocessors x 32) @ 1.15 GHz.,
3 GB memory, connected through PCIe to a quad-core Intel @2.5 GHz.
Sparse Direct Solver and Iterative Refinement

MUMPS package based on multifrontal approach which generates small dense matrix multiplies

**Opteron w/Intel compiler**

- **Iterative Refinement**
- **Single Precision**

**Speedup Over DP**

Tim Davis's Collection, n=100K - 3M
Sparse Iterative Methods (PCG)

**Outer/Inner Iteration**

Outer iterations using 64 bit floating point

- Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$
- for $i = 1, 2, \ldots$
  - solve $Mz^{(i-1)} = r^{(i-1)}$
  - $\rho_{i-1} = r^{(i-1)T}z^{(i-1)}$
  - if $i = 1$
    - $p^{(1)} = z^{(0)}$
  - else
    - $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
    - $p^{(i)} = z^{(i-1)} + \beta_{i-1}p^{(i-1)}$
  - endif
- $q^{(i)} = Ap^{(i)}$
- $\alpha_i = \rho_{i-1}/p^{(i)T}q^{(i)}$
- $x^{(i)} = x^{(i-1)} + \alpha_ip^{(i)}$
- $r^{(i)} = r^{(i-1)} - \alpha_iq^{(i)}$
- check convergence; continue if necessary

Inner iteration:
- In 32 bit floating point

- Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$
- for $i = 1, 2, \ldots$
  - solve $Mz^{(i-1)} = r^{(i-1)}$
  - $\rho_{i-1} = r^{(i-1)T}z^{(i-1)}$
  - if $i = 1$
    - $p^{(1)} = z^{(0)}$
  - else
    - $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
    - $p^{(i)} = z^{(i-1)} + \beta_{i-1}p^{(i-1)}$
  - endif
- $q^{(i)} = Ap^{(i)}$
- $\alpha_i = \rho_{i-1}/p^{(i)T}q^{(i)}$
- $x^{(i)} = x^{(i-1)} + \alpha_ip^{(i)}$
- $r^{(i)} = r^{(i-1)} - \alpha_iq^{(i)}$
- check convergence; continue if necessary

**Outer iteration in 64 bit floating point and inner iteration in 32 bit floating point**
Mixed Precision Computations for Sparse Inner/Outer-type Iterative Solvers

**Speedups** for mixed precision
Inner SP/Outer DP (SP/DP) iter. methods vs DP/DP
(CG$^2$, GMRES$^2$, PCG$^2$, and PGMRES$^2$ with diagonal prec.)
*Higher is better*

**Iterations** for mixed precision
SP/DP iterative methods vs DP/DP
*Lower is better*

Machine:
Intel Woodcrest (3GHz, 1333MHz bus)

Stopping criteria:
Relative to $r_0$ residual reduction ($10^{-12}$)
Intriguing Potential

- Exploit lower precision as much as possible
  - Payoff in performance
    - Faster floating point
    - Less data to move
- Automatically switch between SP and DP to match the desired accuracy
  - Compute solution in SP and then a correction to the solution in DP
- Potential for GPU, FPGA, special purpose processors
  - Use as little you can get away with and improve the accuracy
- Applies to sparse direct and iterative linear systems and Eigenvalue, optimization problems, where Newton’s method is used.

\[ x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \]

Correction = - A\(b - Ax\)
A Call to Action

- Hardware has changed dramatically while software ecosystem has remained stagnant
- Need to exploit new hardware trends (e.g., manycore, heterogeneity) that cannot be handled by existing software stack, memory per socket trends
- Emerging software technologies exist, but have not been fully integrated with system software, e.g., UPC, Cilk, CUDA, HPCS
- Community codes unprepared for sea change in architectures
- No global evaluation of key missing components
International Exascale Software Program

Improve the world’s simulation and modeling capability by improving the coordination and development of the HPC software environment

Workshops:

Build an international plan for coordinating research for the next generation open source software for scientific high-performance computing

www.exascale.org
We believe this needs to be an international collaboration for various reasons including:

• The scale of investment
• The need for international input on requirements
• US, Europeans, Asians, and others are working on their own software that should be part of a larger vision for HPC.
• No global evaluation of key missing components
• Hardware features are uncoordinated with software development
Where We Are Today:

- SC08 (Austin TX) meeting to generate interest
- Funding from DOE’s Office of Science & NSF Office of Cyberinfrastructure and sponsorship by Europeans and Asians
- US meeting (Santa Fe, NM) April 6-8, 2009
  - 65 people
- European meeting (Paris, France) June 28-29, 2009
  - Outline Report
- Asian meeting (Tsukuba Japan) October 18-20, 2009
  - Draft roadmap
  - Refine Report
- SC09 (Portland OR) BOF to inform others
  - Public Comment; Draft Report presented
- European meeting (Oxford, UK) April 13-14, 2010
  - Refine and prioritize roadmap
  - Explore governance structure and management models
- Maui Meeting October 18-19, 2010
- Kobe Meeting - Spring 2011
IESP Executive Committee

- Jack Dongarra, UTK & ORNL
- Pete Beckman, ANL
- Patrick Aerts, NWO Netherlands
- Franck Cappello, INRIA, France
- Thom Dunning, NCSA
- Thomas Lippert, Juelich, Germany
- Satoshi Matsuoka, TiTech, Japan
- Paul Messina, ANL
- Anne Trefethen, Oxford, UK
- Mateo Valero, BSC, Spain
The IESP software roadmap is a planning instrument designed to enable the international HPC community to improve, coordinate and leverage their collective investments and development efforts.

After we determine what needs to be accomplished, our task will be to construct the organizational structures suitable to accomplish the work.
European Exascale Software Initiative - EESI

- A detailed evaluation of how Europe is positioned, its strengths and weaknesses, in the overall international HPC landscape and competition
  - Are European stakeholders willing/able to build an exa-scale prototype/by when?
  - Actors/users/projects

- A European and international vision and roadmap
  - Why is exa-scale initiatives important? Who cares? Impact?
    - Scientific
    - Economic
    - Social benefits

- Dissemination actions
  - Visibility of EESI: who is the potential target public?
    - R&D stakeholders
    - EC and national policy-makers
    - Society as a whole

- Identification of opportunities of worldwide collaborations
  - European position inside IESP: who's doing/deciding what?
  - Contribution to the international dialogs: mutual benefits!

www.exascale.org
EC and G8 Related

- **G8 has a call out for “Interdisciplinary Program on Application Software towards Exascale Computing for Global Scale Issues”**
  - 10 million € over three years
  - An initiative between Research Councils from Canada, France, Germany, Japan, Russia, the UK, and the USA
  - 78 preproposals submitted, 25 selected, expect to fund 6-10
  - Full proposals due August 25th

- **EC FP7: Exascale computing, software and simulation**
  - Announcement due September 28, 2010
  - 25 million €
  - 2 or 3 integrated project to be funded

www.exascale.org
If you are wondering what’s beyond ExaFlops

Mega, Giga, Tera, Peta, Exa, Zetta ...

10^3 kilo
10^6 mega
10^9 giga
10^{12} tera
10^{15} peta
10^{18} exa
10^{21} zetta

10^{24} yotta
10^{27} xona
10^{30} weka
10^{33} vunda
10^{36} uda
10^{39} treda
10^{42} sorta
10^{45} rinta
10^{48} quexa
10^{51} pepta
10^{54} ocha
10^{57} nena
10^{60} minga
10^{63} luma
- www.exascale.org
Summary

- Major Challenges are ahead for extreme computing
  - Power
  - Parallelism
  - Hybrid
  - Fault Tolerance
  - ... and many others not discussed here

- We will need completely new approaches and technologies to reach the Exascale level

- This opens up many new opportunities for applied mathematicians
• “We can only see a short distance ahead, but we can see plenty there that needs to be done.”

  ▪ *Alan Turing (1912–1954)*
Shackleton’s Quote on Exascale

Ernest Shackleton’s 1907 ad in London’s Times, recruiting a crew to sail with him on his exploration of the South Pole

One-Sided Dense Matrix Factorizations
(LU, QR, and Cholesky) from MAGMA

Example: Left-Looking Hybrid Cholesky factorization

MATLAB code | LAPACK code | Hybrid code
--- | --- | ---
(1) \( B = B - A^*A' \) | \( \text{ssyrk(}"L", "N", \&nb, \&j, \&mone, hA(j,0), \ldots \text{)} \) cublas\text{Ssyrk}("L", 'N', nb, j, mone, dA(j,0), ...) | 
\text{cublasGetMatrix}(nb, nb, 4, dA(j, j), *lda, hwork, nb)
(2) \( B = \text{chol}(B, 'lower') \) | \( \text{spotrf(}"L", \&nb, hA(j, j), \text{lda, info)} \) cublas\text{Sgemm}("N", 'T', j, ...) | 
\text{cublasSsyrk("L", 'N', nb, j, mone, dA(j,0), ...)}
(3) \( D = D - C^*A' \) | \text{sgemm("N", 'T', \&j, \ldots \text{)} \) spotrf("L", \&nb, hwork, \&nb, info) | 
\text{cublasGetMatrix}(nb, nb, 4, dA(j, j), *lda)
(4) \( D = B \backslash D \) | \text{strsm("R", 'L', 'T', 'N', \&j, \ldots \text{)} \) \text{cublasStrsm("R", 'L', 'T', 'N', j, \ldots \text{)}} | 
\text{cublasSsyrk("L", 'N', nb, j, mone, dA(j,0), ...)}

CUDA implementation:
- \( \text{a_ref} \) points to the GPU memory
- GPU kernels are started asynchronously which results in overlapping the GPU sgemm with transferring \( T \) to the CPU, factoring it, and sending the result back to the GPU
- For full details see http://www.cs.utk.edu/~tomov/magma/spotrf_gpu.cpp
Communication Reducing Iterative Methods

• Take $k$-steps of Krylov subspace method
  - GMRES, CG, Lanczos, Arnoldi
  - Assume matrix “well-partitioned,” with modest surface-to-volume ratio

• Parallel implementation
  - Conventional: $O(k \log p)$ messages
  - New: $O(\log p)$ messages - optimal

• Serial implementation
  - Conventional: $O(k)$ moves of data from slow to fast memory
  - New: $O(1)$ moves of data - optimal

• Can incorporate some preconditioners
  - Need to be able to “compress” interactions between distant $i, j$
  - Hierarchical, semiseparable matrices ...

• Lots of speed up possible (modeled and measured)
  - Price: some redundant computation
Minimizing Communication of GMRES to solve $Ax=b$

- **GMRES**: find $x$ in $\text{span}\{b, Ab, \ldots, A^k b\}$ minimizing $||Ax-b||_2$
- **Cost of $k$ steps of standard GMRES vs new GMRES**

**Standard GMRES**

for $i=1$ to $k$

$w = A \cdot v(i-1)$

$\text{MGS}(w, v(0), \ldots, v(i-1))$

update $v(i), H$

endfor

solve LSQ problem with $H$

**Communication-avoiding GMRES**

$W = [v, Av, A^2v, \ldots, A^k v]$

$[Q,R] = \text{TSQR}(W)$ ... “Tall Skinny QR”

Build $H$ from $R$, solve LSQ problem

Sequential: #words_moved = $O(k \cdot \text{nnz})$ from SpMV

+ $O(k^2 \cdot n)$ from MGS

Parallel: #messages = $O(k)$ from SpMV

+ $O(k^2 \cdot \log p)$ from MGS

Sequential: #words_moved = $O(\text{nnz})$ from SpMV

+ $O(k \cdot n)$ from TSQR

Parallel: #messages = $O(1)$ from computing $W$

+ $O(\log p)$ from TSQR

**Numerical issue with potential loss of precision from computing $W$ from power method.**
<table>
<thead>
<tr>
<th>Rank</th>
<th>Site</th>
<th>Manufact.</th>
<th>Computer</th>
<th>Cores</th>
<th>Gflop/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>70</td>
<td>CINECA</td>
<td>IBM</td>
<td>Power 575, p6 4.7 GHz, Infiniband</td>
<td>5376</td>
<td>78680</td>
</tr>
<tr>
<td>129</td>
<td>Telecom</td>
<td>IBM</td>
<td>BladeCenter HS22 Cluster, Xeon QC GT 2.53 GHz, GigEthernet</td>
<td>8048</td>
<td>45528</td>
</tr>
<tr>
<td>211</td>
<td>CILEA</td>
<td>HP</td>
<td>Cluster Platform 3000 BL2x220, X56xx 3.0 GHz, Infiniband QDR</td>
<td>4032</td>
<td>35665</td>
</tr>
<tr>
<td></td>
<td>Energy Company (A)</td>
<td>IBM</td>
<td>BladeCenter HS22 Cluster, Xeon QC X56xx 2.66 GHz, Infiniband</td>
<td>3408</td>
<td>31310</td>
</tr>
<tr>
<td>296</td>
<td>Energy Company (A)</td>
<td>IBM</td>
<td>BladeCenter HS22 Cluster, Xeon QC X56xx 2.66 GHz, Infiniband</td>
<td>3408</td>
<td>31310</td>
</tr>
<tr>
<td>297</td>
<td>Energy Company (A)</td>
<td>IBM</td>
<td>BladeCenter HS22 Cluster, Xeon QC X56xx 2.66 GHz, Infiniband</td>
<td>3408</td>
<td>31310</td>
</tr>
<tr>
<td>404</td>
<td>Sardegna Ricerche</td>
<td>HP</td>
<td>Cluster Platform 3000 BL460c G1, Xeon E5440 2.83 GHz, Infiniband</td>
<td>3088</td>
<td>27708</td>
</tr>
</tbody>
</table>
DP Cholesky with Multiple GPUs

Speed up of the hybrid DPOTFF (4 Opteron 1.8GHz and 4 GPU TESLA C1060 1.44GHz)

- Red: PLASMA-MAGMA 4 CPUs-4 GPUs
- Green: PLASMA-MAGMA 3 CPUs-3 GPUs
- Blue: PLASMA-MAGMA 2 CPUs-2 GPUs
- Black: PLASMA-MAGMA 1 CPU-1 GPU

Gflop/s vs Matrix Size
How to Code for GPUs?

- **Complex question**
  - Language, programming model, user productivity, etc

- **Recommendations**
  - **Use CUDA / OpenCL**
    [already demonstrated benefits in many areas; data-based parallelism; move to support task-based]
  - **Use GPU BLAS**
    [high level; available after introduction of shared memory - can do data reuse; leverage existing developments]
  - **Use Hybrid Algorithms**
    [currently GPUs - massive parallelism but serial kernel execution; hybrid approach - small non-parallelizable tasks on the CPU, large parallelizable tasks on the GPU]
Evolution of GPUs

GPUs: excelling in graphics rendering

- Scene model → Streams of data → Graphics pipelined computation → Final image

Repeated fast over and over: e.g. TV refresh rate is 30 fps; limit is 60 fps

- Requires enormous computational power
- Allows for high parallelism
- Needs high bandwidth vs low latency
  (as low latencies can be compensated with deep graphics pipeline)

Obviously, this pattern of computation is common with many other applications
Moore’s Law Reinterpreted

- Number of cores per chip doubles every 2 year, while clock speed decreases (not increases).
  - Need to deal with systems with millions of concurrent threads
    - Future generation will have billions of threads!
  - Need to be able to easily replace inter-chip parallelism with intro-chip parallelism
- Number of threads of execution doubles every 2 year
## Potential System Architectures

<table>
<thead>
<tr>
<th>Systems</th>
<th>2009</th>
</tr>
</thead>
<tbody>
<tr>
<td>System peak</td>
<td>2 Pflop/s</td>
</tr>
<tr>
<td>System memory</td>
<td>0.3 PB</td>
</tr>
<tr>
<td>Node performance</td>
<td>125 Gflop/s</td>
</tr>
<tr>
<td>Node memory BW</td>
<td>25 GB/s</td>
</tr>
<tr>
<td>Node concurrency</td>
<td>12</td>
</tr>
<tr>
<td>Interconnect BW</td>
<td>1.5 GB/s</td>
</tr>
<tr>
<td>System size (nodes)</td>
<td>18,700</td>
</tr>
<tr>
<td>Total concurrency</td>
<td>225,000</td>
</tr>
<tr>
<td>Storage</td>
<td>15 PB</td>
</tr>
<tr>
<td>IO</td>
<td>0.2 TB/s</td>
</tr>
<tr>
<td>MTTI</td>
<td>days</td>
</tr>
<tr>
<td>Power</td>
<td>7 MW</td>
</tr>
</tbody>
</table>
Conclusions

- For the last decade or more, the research investment strategy has been overwhelmingly biased in favor of hardware.
- This strategy needs to be rebalanced - barriers to progress are increasingly on the software side.
- Moreover, the return on investment is more favorable to software.
  - Hardware has a half-life measured in years, while software has a half-life measured in decades.
- High Performance Ecosystem out of balance
  - Hardware, OS, Compilers, Software, Algorithms, Applications
    - No Moore’s Law for software, algorithms and applications
Collaborators / Support

Employment opportunities for post-docs in the PLASMA/MAGMA projects

**PLASMA** Parallel Linear Algebra Software for Multicore Architectures
http://icl.cs.utk.edu/plasma/

**MAGMA** Matrix Algebra on GPU and Multicore Architectures
http://icl.cs.utk.edu/magma/

Emmanuel Agullo, Jim Demmel, Jack Dongarra, Bilel Hadri, Jakub Kurzak, Julie & Julien Langou, Hatem Ltaief, Piotr Luszczek, Stan Tomov
- **Strong scaling**: fixed problem size.
  - Data on each node decreases as the number of nodes increases.

- **Weak scaling**: fixed the data size on each node.
  - Problem size increases as the number of node increases.
Review: two definitions of scalability

“Strong scaling”
- execution time decreases in inverse proportion to the number of processors
- fixed size problem overall
- often instead graphed as reciprocal, “speedup”

“Weak scaling”
- execution time remains constant, as problem size and processor number are increased in proportion
- fixed size problem per processor
- Various sub-types of weak-scaling “memory bound”, etc. (see Kumar et
Symmetric Positive Definite

Cholesky-GPU
Intel (R) Xeon (R) E5410 2.33 GHz (8 Core)
GForce GTX 280 1.3 GHz (240 Core)

<table>
<thead>
<tr>
<th></th>
<th>Single Prec</th>
<th>Double Prec</th>
<th>Iter Refine</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU</td>
<td>NVIDIA GeForce GTX 280</td>
<td>CUBLAS 2.2, s/dgemm peak: 375 / 75 GFlop/s</td>
<td></td>
</tr>
<tr>
<td>CPU</td>
<td>Intel Xeon dual socket quad-core @2.33 GHz</td>
<td>MKL 10.0, s/dgemm peak: 17.5 / 8.6 GFlop/s</td>
<td></td>
</tr>
</tbody>
</table>

GFlops/s vs Dimension

GPU BLAS
CPU BLAS
• **Projections**
  - Performance
  - Memory

• **Async**
  - Break fork-join
  - DAGs
  - New algorithms - numerical issues
  - Communication avoiding
  - Chaotic iteration

• **Mixed precision**
  - Iter refine
  - precond

• **Hybrid**
  - balance
  - autotune

• **FT**
  - Number of approaches
MAGMA Software

- Available through MAGMA's homepage
- Included are the 3 one-sided matrix factorizations
- Iterative Refinement Algorithm (Mixed Precision)
- Standard (LAPACK) data layout and accuracy
- Two LAPACK-style interfaces
  - CPU interface: both input and output are on the CPU
  - GPU interface: both input and output are on the GPU
- This release is intended for single GPU
Purpose

• The IESP software roadmap is a planning instrument designed to enable the international HPC community to improve, coordinate and leverage their collective investments and development efforts.

• After we determine what needs to be accomplished, our task will be to construct the organizational structures suitable to accomplish the work.
Diskless Checkpointing 1/2

Principle: Compute a checksum of the processes’ memory and store it on spare processors.

Advantage: does not require ckpt on stable storage.

4 computing processors

Add fifth “non computing” processor

A) Every process saves a copy of its local state of in memory or local disc

Start the computation

Perform a checkpoint

Continue the computation

B) Perform a global bitstream or floating point operation on all saved local states

All processes restore its local state from the one saved in memory or local disc

Failure

Ready for recovery

Recover P2 data
Diskless Checkpointing 2/2

• Could be done at application and system levels

• Process data could be considered (and encoded) either as bit-streams or as floating point numbers. Computing the checksum from bit-streams uses operations such as parity. Computing checksum from floating point numbers uses operations such as addition.

• Can survive multiple failures of arbitrary patterns. Reed Solomon for bit-streams and weighted checksum for floating point numbers (sensitive to round-off errors).

• Work with incremental ckpt.

• Need spare nodes and double the memory occupation (to survive failures during ckpt.) --> increases the overall cost and #failures

• Need coordinated checkpointing or message logging protocol

• Need very fast encoding & reduction operations

• Need automatic Ckpt protocol or program modifications

Challenge: experiment more Diskless CKPT and in very large machines (current result are for ~1000 CPUs)
In 1984, Huang and Abraham, proposed the ABFT to detect and correct errors in some matrix operations on systolic arrays.

ABFT encodes data & redesign algo. to operate on encoded data. Failure are detected and corrected off-line (after execution).

ABFT variation for on-line recovery (runtime detects failures + robust to failures):

• Similar to Diskless ckpt., an extra processor is added, Pi+1, store the checksum of data:
  (vector X and Y in this case)

  \[ X_c = X_1 + \ldots + X_p, \quad Y_c = Y_1 + \ldots + Y_p. \]

  \[ X_f = [X_1, \ldots X_p, X_c], \quad Y_f = [Y_1, \ldots Y_p, Y_c], \]

• Operations are performed on Xf and Yf instead of X and Y: Zf=Yf+Zf

• Compared to diskless checkpointing, the memory AND CPU of Pc take part of the computation):
  • No global operation for Checksum!
  • No local checkpoint!

Works for many Linear Algebra operations:

Matrix Multiplication: \( A \times B = C \rightarrow A_c \times B_r = C_f \)
LU Decomposition: \( C = L \times U \rightarrow C_f = L_c \times U_r \)
Addition: \( A + B = C \rightarrow A_f + B_f = C_f \)
Scalar Multiplication: \( c \times A_f = (c \times A)f \)
Transpose: \( A_f^T = (AT)f \)
Cholesky factorization & QR factorization
“Naturally fault tolerant algorithm”

Natural fault tolerance is the ability to tolerate failures through the mathematical properties of the algorithm itself, without requiring notification or recovery.

The algorithm includes natural compensation for the lost information.

For example, an iterative algorithm may require more iterations to converge, but it still converges despite lost information.

Assumes that a maximum of 0.1% of tasks may fail.

Ex1: Meshless iterative methods + chaotic relaxation (asynchronous iterative methods).

This algorithm share some features with SelfStabilization algorithms: detection of termination is very hard! → it provides the max « eventually »… BUT, it does not tolerate Byzantine faults (SelfStabilization does for transient failures + acyclic topology).
Proactive Migration

- **Principle:** predict failures and migrate processes before failures

- Prediction models are based on the analysis of correlations between non-fatal and fatal errors, and temporal and spatial correlations between failure events.

- Results on the *100 first days of BlueGene/L demonstrate good failure predictability*: 50% of I/O failures could have been predicted (based on trace analysis).

  Note that Memory failures are much less predictable!

- Bad prediction has a cost (false positives and negatives have an impact on performance) → false negatives impose to use rollback-recovery.

- Migration has a cost (need to checkpoint and log or delay messages)

- What to migrate?
  - Virtual Machine, Process checkpoint?
  - Only application state (user checkpoint)?

- Challenge: Analyze more traces, Identify more correlations, Improve predictive algorithms
Fault Recovery Options

- **Saved State**
  - Restart – from checkpoint file
  - Restart from **local** checkpoint
  - Recalculate lost data from in-memory checkpoint (RAID like)

- **No Checkpoint**
  - Lossy recalculation of lost data
  - Recalculate lost data from initial and remaining data
  - Replicate computation across system
  - Reassign lost work to another resource
  - Use natural fault tolerant algorithms
Fault Tolerance

**Hard errors** – permanent component failure either HW or SW (hung or crash)

**Soft errors** – transient errors, a blip or short term failure of either HW or SW

**Silent errors** – undetected errors either hard or soft, due to lack of detectors for a component or inability to detect (transient effect too short). **Real danger is that answer may be incorrect but the user wouldn’t know.**

**HW (node and interconnect )resilience needed to reduce Silent errors** – Either turn them into Hard or Soft errors or fix them
## Potential System Architecture

<table>
<thead>
<tr>
<th>Systems</th>
<th>2009</th>
<th>2018</th>
</tr>
</thead>
<tbody>
<tr>
<td>System peak</td>
<td>2 Pflop/s</td>
<td>1 Eflop/s</td>
</tr>
<tr>
<td>Power</td>
<td>6 MW</td>
<td>~20 MW</td>
</tr>
<tr>
<td>System memory</td>
<td>0.3 PB</td>
<td>32 - 64 PB</td>
</tr>
<tr>
<td>Node performance</td>
<td>125 GF</td>
<td>1,2 or 15TF</td>
</tr>
<tr>
<td>Node memory BW</td>
<td>25 GB/s</td>
<td>2-4TB/s</td>
</tr>
<tr>
<td>Node concurrency</td>
<td>12</td>
<td>O(1k) or 10k</td>
</tr>
<tr>
<td>Total Node Interconnect BW</td>
<td>3.5 GB/s</td>
<td>200-400GB/s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(1:4 or 1:8 from memory BW)</td>
</tr>
<tr>
<td>System size (nodes)</td>
<td>18,700</td>
<td>O(100,000) or O(1M)</td>
</tr>
<tr>
<td>Total concurrency</td>
<td>225,000</td>
<td>O(billion) [O(10) to O(100) for latency hiding]</td>
</tr>
<tr>
<td>Storage</td>
<td>15 PB</td>
<td>500-1000 PB (&gt;10x system memory is min)</td>
</tr>
<tr>
<td>IO</td>
<td>0.2 TB</td>
<td>60 TB/s (how long to drain the machine)</td>
</tr>
<tr>
<td>MTTI</td>
<td>days</td>
<td>O(0.1 day)</td>
</tr>
</tbody>
</table>
Communication Reducing QR Factorization

TS matrix
- MT=6 and NT=3
- split into 2 domains

3 overlapped steps
- panel factorization
- updating the trailing submatrix
- merge the domains

August 28, 2009
TS matrix

- MT=6 and NT=3
- split into 2 domains

3 overlapped steps

- **panel factorization**
  - updating the trailing submatrix
  - merge the domains

Communication Reducing QR Factorization

August 28, 2009
Communication Reducing QR Factorization

- MT=6 and NT=3
- split into 2 domains

3 overlapped steps

- panel factorization
- updating the trailing submatrix

- merge the domains

August 28, 2009
Communication Reducing QR Factorization

**TS matrix**
- MT=6 and NT=3
- split into 2 domains

3 overlapped steps

- **panel factorization**
- updating the trailing submatrix

- merge the domains

---

**August 28, 2009**
TS matrix

- MT = 6 and NT = 3
- split into 2 domains

3 overlapped steps

- panel factorization
- updating the trailing submatrix
- merge the domains

August 28, 2009
Communication Reducing QR Factorization

TS matrix
- MT=6 and NT=3
- split into 2 domains

3 overlapped steps
- panel factorization
- updating the trailing submatrix
- merge the domains

August 28, 2009
Communication Reducing QR Factorization

- TS matrix
  - MT=6 and NT=3
  - split into 2 domains

3 overlapped steps

- panel factorization
- updating the trailing submatrix
- merge the domains

August 28, 2009
Communication Reducing QR Factorization

TS matrix
- $M_1=6$ and $N_1=3$
- split into 2 domains

3 overlapped steps
- panel factorization
- updating the trailing submatrix
- merge the domains

August 28, 2009
Communication Reducing QR Factorization

- TS matrix
  - MT=6 and NT=3
  - split into 2 domains

- 3 overlapped steps
  - panel factorization
  - updating the trailing submatrix
  - merge the domains
### Communication Reducing QR Factorization

**TS matrix**
- $MT=6$ and $NT=3$
- split into 2 domains

3 overlapped steps
- panel factorization
- updating the trailing submatrix
- **merge the domains**

![TS matrix diagram]

August 28, 2009
Communication Reducing QR Factorization

- TS matrix
  - MT=6 and NT=3
  - split into 2 domains

- 3 overlapped steps
  - panel factorization
    - updating the trailing submatrix
    - merge the domains
TS matrix
- MT=6 and NT=3
- split into 2 domains

3 overlapped steps
- **panel factorization**
  - updating the trailing submatrix
  - merge the domains
### TS matrix

- MT = 6 and NT = 3
- split into 2 domains

3 overlapped steps

- **panel factorization**
  - updating the trailing submatrix
  - merge the domains

![Matrix Diagram](attachment:image.png)

August 28, 2009
TS matrix

- MT=6 and NT=3
- split into 2 domains

3 overlapped steps

- panel factorization
- updating the trailing submatrix

- merge the domains

August 28, 2009
Communication Reducing QR Factorization

- TS matrix
  - MT=6 and NT=3
  - split into 2 domains

- 3 overlapped steps
  - panel factorization
  - updating the trailing submatrix
  - merge the domains

- Final R computed

August 28, 2009
# 35rd List: The TOP10

<table>
<thead>
<tr>
<th>Rank</th>
<th>Site</th>
<th>Computer</th>
<th>Country</th>
<th>Cores [Pflops]</th>
<th>% of Peak</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DOE / OS Oak Ridge Nat Lab</td>
<td>Jaguar / Cray Cray XT5sixCore 2.6 GHz</td>
<td>USA</td>
<td>224,162</td>
<td>1.76</td>
</tr>
<tr>
<td>2</td>
<td>Nat. Supercomputer Center in Shenzhen</td>
<td>Nebulea / Dawning / TC3600 Blade, Intel X5650, Nvidia C2050 GPU</td>
<td>China</td>
<td>120,640</td>
<td>1.27</td>
</tr>
<tr>
<td>3</td>
<td>DOE / NNSA Los Alamos Nat Lab</td>
<td>Roadrunner / IBM BladeCenterQS22/LS21</td>
<td>USA</td>
<td>122,400</td>
<td>1.04</td>
</tr>
<tr>
<td>4</td>
<td>NSF / NICS / U of Tennessee</td>
<td>Kraken / Cray Cray XT5sixCore 2.6 GHz</td>
<td>USA</td>
<td>98,928</td>
<td>.831</td>
</tr>
<tr>
<td>5</td>
<td>Forschungszentrum Jueltich (FZJ)</td>
<td>Jugene / IBM Blue Gene/P Solution</td>
<td>Germany</td>
<td>294,912</td>
<td>.825</td>
</tr>
<tr>
<td>6</td>
<td>NASA / Ames Research Center/NAS</td>
<td>Pleiades / SGI SGI Altix ICE 8200EX</td>
<td>USA</td>
<td>56,320</td>
<td>.544</td>
</tr>
<tr>
<td>7</td>
<td>National SC Center in Tianjin / NUDT</td>
<td>Tianhe-1 / NUDT TH-1 / IntelQC + AMD ATI Radeon 4870</td>
<td>China</td>
<td>71,680</td>
<td>.563</td>
</tr>
<tr>
<td>8</td>
<td>DOE / NNSA Lawrence Livermore NL</td>
<td>BlueGene/L IBM eServerBlue Gene Solution</td>
<td>USA</td>
<td>212,992</td>
<td>.478</td>
</tr>
<tr>
<td>9</td>
<td>DOE / OS Argonne Nat Lab</td>
<td>Intrepid / IBM Blue Gene/P Solution</td>
<td>USA</td>
<td>163,840</td>
<td>.458</td>
</tr>
<tr>
<td>10</td>
<td>DOE / NNSA Sandia Nat Lab</td>
<td>Red Sky / Sun / 6275 SunBlade</td>
<td>USA</td>
<td>42,440</td>
<td>.433</td>
</tr>
</tbody>
</table>
# 35th List: The TOP10

<table>
<thead>
<tr>
<th>Rank</th>
<th>Site</th>
<th>Computer</th>
<th>Country</th>
<th>Cores</th>
<th>Rmax [Pflops]</th>
<th>% of Peak</th>
<th>Power [MW]</th>
<th>MFlops/Watt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DOE / OS Oak Ridge Nat Lab</td>
<td>Jaguar / Cray Cray XT5sixCore 2.6 GHz</td>
<td>USA</td>
<td>224,162</td>
<td>1.76</td>
<td>75</td>
<td>7.0</td>
<td>251</td>
</tr>
<tr>
<td>2</td>
<td>Nat. Supercomputer Center in Shenzhen</td>
<td>Nebulea / Dawning / TC3600 Blade, Intel X5650, Nvidia C2050 GPU</td>
<td>China</td>
<td>120,640</td>
<td>1.27</td>
<td>43</td>
<td>2.58</td>
<td>493</td>
</tr>
<tr>
<td>3</td>
<td>DOE / NNSA Los Alamos Nat Lab</td>
<td>Roadrunner / IBM BladeCenterQS22/LS21</td>
<td>USA</td>
<td>122,400</td>
<td>1.04</td>
<td>76</td>
<td>2.48</td>
<td>446</td>
</tr>
<tr>
<td>4</td>
<td>NSF / NICS / U of Tennessee</td>
<td>Kraken/ Cray Cray XT5sixCore 2.6 GHz</td>
<td>USA</td>
<td>98,928</td>
<td>.831</td>
<td>81</td>
<td>3.09</td>
<td>269</td>
</tr>
<tr>
<td>5</td>
<td>ForschungszentrumJuelich (FZJ)</td>
<td>Jugene / IBM Blue Gene/P Solution</td>
<td>Germany</td>
<td>294,912</td>
<td>.825</td>
<td>82</td>
<td>2.26</td>
<td>365</td>
</tr>
<tr>
<td>6</td>
<td>NASA / Ames Research Center/NAS</td>
<td>Pleiades / SGI SGI Altix ICE 8200EX</td>
<td>USA</td>
<td>56,320</td>
<td>.544</td>
<td>82</td>
<td>3.1</td>
<td>175</td>
</tr>
<tr>
<td>7</td>
<td>National SC Center in Tianjin / NUDT</td>
<td>Tianhe-1 / NUDT TH-1 / IntelQC + AMD ATI Radeon 4870</td>
<td>China</td>
<td>71,680</td>
<td>.563</td>
<td>46</td>
<td>1.48</td>
<td>380</td>
</tr>
<tr>
<td>8</td>
<td>DOE / NNSA Lawrence Livermore NL</td>
<td>BlueGene/L IBM eServerBlue Gene Solution</td>
<td>USA</td>
<td>212,992</td>
<td>.478</td>
<td>80</td>
<td>2.32</td>
<td>206</td>
</tr>
<tr>
<td>9</td>
<td>DOE / OS Argonne Nat Lab</td>
<td>Intrepid / IBM Blue Gene/P Solution</td>
<td>USA</td>
<td>163,840</td>
<td>.458</td>
<td>82</td>
<td>1.26</td>
<td>363</td>
</tr>
<tr>
<td>10</td>
<td>DOE / NNSA Sandia Nat Lab</td>
<td>Red Sky / Sun / 6275 SunBlade</td>
<td>USA</td>
<td>42,440</td>
<td>.433</td>
<td>87</td>
<td>2.4</td>
<td>180</td>
</tr>
</tbody>
</table>
Parallel Tasks in LU

- Break into smaller tasks and remove dependencies
LU Factorization in Double Precision

Matrix Size vs. GFlop/s

- **FERMI MAGMA**
  - Tesla C2050: 448 CUDA cores @ 1.15GHz
  - SP/DP peak is 1030 / 515 GFlop/s

- **INSTANBUL PLASMA**
  - AMD 8 socket 6 core (48 cores) @2.8GHz
  - SP/DP peak is 1075 / 538 GFlop/s

- **GTX 280 MAGMA**
Parallel Tasks in LU/LL$^T$/QR

- Break into smaller tasks and remove dependencies
- Tile LU factorization on a square matrix with 5 x 5 tiles. Each tile is of size b x b and corresponds to a fine grain task. The arcs show the data dependencies between the tasks.

* LU does block pair wise pivoting
#3 LANL Roadrunner
A Petascale System in 2008

“Connected Unit” cluster
192 Opteron nodes
(180 w/ 2 dual-Cell blades connected w/ 4 PCIe x8 links)

≈ 13,000 Cell HPC chips
≈ 1.33 PetaFlop/s (from Cell)
≈ 7,000 dual-core Opterons
≈ 122,000 cores

17 clusters

2nd stage InfiniBand 4x DDR interconnect
(18 sets of 12 links to 8 switches)

2nd stage InfiniBand interconnect (8 switches)

Based on the 100 Gflop/s (DP) Cell chip

Hybrid Design (2 kinds of chips & 3 kinds of cores)
Programming required at 3 levels.

Dual Core Opteron Chip