The Impact of Multicore on Mathematical Software

Jack Dongarra
INNOVATIVE COMPUTING LABORATORY
University of Tennessee
Oak Ridge National Laboratory
Outline

• Top500 Results
• Four Important Concepts that Will Effect Math Software
  ▪ Effective Use of Many-Core
  ▪ Exploiting Mixed Precision in Our Numerical Computations
  ▪ Self Adapting / Auto Tuning of Software
  ▪ Fault Tolerant Algorithms
- Listing of the 500 most powerful Computers in the World
- Yardstick: $R_{\text{max}}$ from LINPACK MPP
  \[ Ax=b, \text{ dense problem} \]
- 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

- NEC Earth Simulator: 281 TF/s
- IBM BlueGene/L: 1.17 TF/s
- Intel ASCI White: 4.0 TF/s
- Fujitsu 'NWT': 1 Gflop/s
- My Laptop: 0.4 GF/s

Timeline:
- 1993: 100 Mflop/s
- 1995: 1 Gflop/s
- 1997: 10 Gflop/s
- 1999: 100 Gflop/s
- 2001: 1 Tflop/s
- 2003: 10 Tflop/s
- 2005: 100 Tflop/s
- 2007: 1 Pflop/s

Growth rates:
- 1993 to 1995: 0.4 GF/s
- 1995 to 1997: 59.7 GF/s

Future projections:
- 6-8 years: 1 Gflop/s
- N=500
- N=1
Performance Projection

- 1 TF/s
- 100 GF/s
- 10 TF/s
- 100 TF/s
- 1 PF/s
- 100 PF/s
- 10 PF/s
- 1 EF/s

SUM

N=1

N=500

1993 1995 1997 1999 2001 2003 2005 2007 2009 2011 2013 2015
Chips Used in Each of the 500 Systems

- Intel EM64T: 46%
- Intel IA-32: 6%
- AMD x86_64: 21%
- IBM Power: 17%
- Intel IA-64: 6%
- Cray: 0%
- HP Alpha: 0%
- HP PA-RISC: 2%
- NEC: 1%
- Sun Sparc: 1%

96% = 58% Intel
17% IBM
21% AMD
GigE + Infiniband + Myrinet = 76%
Increasing the number of gates into a tight knot and decreasing the cycle time of the processor

Power \propto \text{Frequency}^3

We have seen increasing number of gates on a chip and increasing clock speed.

Heat becoming an unmanageable problem, Intel Processors > 100 Watts

We will not see the dramatic increases in clock speeds in the future.

However, the number of gates on a chip will continue to increase.
80 Core

- Intel’s 80 Core chip
  - 12 Tflop/s
  - 62 Watts
  - 1.2 TB/s internal BW

Intel Prototype May Herald a New Age of Processing

By JOHN MARKOFF
Published, February 12, 2007

SAN FRANCISCO, Feb. 11 — Intel will demonstrate on Monday an experimental computer chip with 80 separate processing engines, or cores, that company executives say provides a model for commercial chips that will be used widely in standard desktop, laptop and server computers within five years.

The new processor, which the company first described as a Teraflop Chip at a conference last year, will be detailed in a technical paper to be presented on the opening day of the International Solid States Circuits Conference, beginning here on Monday.

While the chip is not compatible with Intel’s current chips, the company said it had already begun design work on a commercial version that would essentially have dozens or even hundreds of Intel-compatible microprocessors laid out in a tiled pattern on a single chip.
What's Next?

- All Large Core
- Mixed Large and Small Core
- Many Small Cores
- Many Floating-Point Cores
- + 3D Stacked Memory

Different Classes of Chips:
- Home
- Games / Graphics
- Business
- Scientific

SRAM
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
### A New Generation of Software:

**Algorithms follow hardware evolution in time**

<table>
<thead>
<tr>
<th>LINPACK (80’s)</th>
<th>Rely on Level-1 BLAS operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Vector operations)</td>
<td></td>
</tr>
<tr>
<td><strong>LAPACK (90’s)</strong></td>
<td>Rely on Level-3 BLAS operations</td>
</tr>
<tr>
<td>(Blocking, cache friendly)</td>
<td></td>
</tr>
</tbody>
</table>

Those new algorithms:
- have a very low granularity, they scale very well (multicore, petascale computing, …)
- remove a lot of dependencies among the tasks, (multicore, distributed computing)
- avoid latency (distributed computing, out of core)
- rely on fast kernels

Those new algorithms need new kernels and rely on efficient scheduling algorithms.
A New Generation of Software: Parallel Linear Algebra Software for Multicore Architectures (PLASMA)

Algorithms follow hardware evolution in time

<table>
<thead>
<tr>
<th>Software</th>
<th>Algorithms</th>
<th>Rely on</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINPACK (80's)</td>
<td>(Vector operations)</td>
<td>Level-1 BLAS operations</td>
</tr>
<tr>
<td>LAPACK (90's)</td>
<td>(Blocking, cache friendly)</td>
<td>Level-3 BLAS operations</td>
</tr>
<tr>
<td>PLASMA (00's)</td>
<td>New Algorithms (many-core friendly)</td>
<td>a DAG/scheduler, block data layout, some extra extra kernels</td>
</tr>
</tbody>
</table>

Those new algorithms
- have a very low granularity, they scale very well (multicore, petascale computing, … )
- removes a lots of dependencies among the tasks, (multicore, distributed computing)
- avoid latency (distributed computing, out-of-core)
- rely on fast kernels

Those new algorithms need new kernels and rely on efficient scheduling algorithms.
Steps in the LAPACK LU

DGETF2
(Factor a panel)

DLSWP
(Backward swap)

DLSWP
(Forward swap)

DTRSM
(Triangular solve)

DGEMM
(Matrix multiply)

LAPACK

LAPACK

LAPACK

BLAS

BLAS
LU Timing Profile (4 processor system)

Threads – no lookahead

Time for each component

1D decomposition and SGI Origin

DGETF2
DLASWP(L)
DLASWP(R)
DTRSM
DGEMM

Bulk Sync Phases
Event Driven Multithreading

while(1)
    fetch_task();
    switch(task.type) {
        case PANEL:
            dgetf2();
            update_progress();
        case COLUMN:
            dlaswp();
            dtrsm();
            dgemm();
            update_progress();
        case END:
            for()
                dlaswp();
            return;
    }

Reorganizing algorithms to use this approach
Fork-Join vs. Dynamic Execution

Experiments on
Intel’s Quad Core Clovertown
with 2 Sockets w/ 8 Treads
Fork-Join vs. Dynamic Execution

Experiments on Intel’s Quad Core Clovertown with 2 Sockets w/ 8 Treads

Fork-Join – parallel BLAS

DAG-based – dynamic scheduling
With the Hype on Cell & PS3 We Became Interested

- The PlayStation 3's CPU based on a "Cell" processor
- Each Cell contains a Power PC processor and 8 SPEs. (SPE is processing unit, SPE: SPU + DMA engine)
  - An SPE is a self contained vector processor which acts independently from the others.
    - 4 way SIMD floating point units capable of a total of 25.6 Gflop/s @ 3.2 GHZ
  - 204.8 Gflop/s peak!
  - The catch is that this is for 32 bit floating point; (Single Precision SP)
  - And 64 bit floating point runs at 14.6 Gflop/s total for all 8 SPEs!!
    - Divide SP peak by 14; factor of 2 because of DP and 7 because of latency issues

SPE ~ 25 Gflop/s peak
Performance of Single Precision on Conventional Processors

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

- The Intel Pentium and AMD Opteron have SSE2
  - 2 flops/cycle DP
  - 4 flops/cycle SP

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

<table>
<thead>
<tr>
<th>Processor</th>
<th>Size</th>
<th>SGEMM/ DGEMM</th>
<th>Size</th>
<th>SGEMV/ DGEMV</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD Opteron 246</td>
<td>3000</td>
<td>2.00</td>
<td>5000</td>
<td>1.70</td>
</tr>
<tr>
<td>UltraSparc-Ile</td>
<td>3000</td>
<td>1.64</td>
<td>5000</td>
<td>1.66</td>
</tr>
<tr>
<td>Intel PIII Coppermine</td>
<td>3000</td>
<td>2.03</td>
<td>5000</td>
<td>2.09</td>
</tr>
<tr>
<td>PowerPC 970</td>
<td>3000</td>
<td>2.04</td>
<td>5000</td>
<td>1.44</td>
</tr>
<tr>
<td>Intel Woodcrest</td>
<td>3000</td>
<td>1.81</td>
<td>5000</td>
<td>2.18</td>
</tr>
<tr>
<td>Intel XEON</td>
<td>3000</td>
<td>2.04</td>
<td>5000</td>
<td>1.82</td>
</tr>
<tr>
<td>Intel Centrino Duo</td>
<td>3000</td>
<td>2.71</td>
<td>5000</td>
<td>2.21</td>
</tr>
</tbody>
</table>

Single precision is faster because:
- Higher parallelism in SSE/vector units
- Reduced data motion
- Higher locality in cache
32 or 64 bit Floating Point Precision?

- A long time ago 32 bit floating point was used
  - Still used in scientific apps but limited
- Most apps use 64 bit floating point
  - Accumulation of round off error
    - A 10 TFlop/s computer running for 4 hours performs > 1 Exaflop \((10^{18})\) ops.
  - Ill conditioned problems
  - IEEE SP exponent bits too few (8 bits, \(10^{\pm 38}\))
  - Critical sections need higher precision
    - Sometimes need extended precision (128 bit fl pt)
  - However some can get by with 32 bit fl pt in some parts
- Mixed precision a possibility
  - Approximate in lower precision and then refine or improve solution to high precision.
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.

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

WHILE $\| r \|$ not small enough

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

END
Mixed-Precision Iterative Refinement

- Iterative refinement for dense systems, \( Ax = b \), can work this way.

\[
\begin{align*}
L U &= lu(A) \quad \text{SINGLE} \quad O(n^3) \\
x &= L\backslash(U\backslash b) \quad \text{SINGLE} \quad O(n^2) \\
r &= b - Ax \quad \text{DOUBLE} \quad O(n^2) \\
\text{WHILE } || r || \text{ not small enough} \\
z &= L\backslash(U\backslash r) \quad \text{SINGLE} \quad O(n^2) \\
x &= x + z \quad \text{DOUBLE} \quad O(n^1) \\
r &= b - Ax \quad \text{DOUBLE} \quad O(n^2) \\
\end{align*}
\]

- 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 Multiple Precision Iterative Refinement

New routines in LAPACK that do this for LU and $LL^T$
What about the Cell?

- **Power PC at 3.2 GHz**
  - DGEMM at 5 Gflop/s
  - Altivec peak at 25.6 Gflop/s
    - Achieved 10 Gflop/s SGEMM

- **8 SPUs**
  - 204.8 Gflop/s peak!
  - The catch is that this is for 32 bit floating point; (Single Precision SP)
  - And 64 bit floating point runs at 14.6 Gflop/s total for all 8 SPEs!!
    - Divide SP peak by 14; factor of 2 because of DP and 7 because of latency issues
IBM Cell 3.2 GHz, $Ax = b$

- 8 SGEMM (Embarrassingly Parallel)

Graph showing performance comparison of SP and DP operations with IBM Cell 3.2 GHz processor. The graph plots GFlop/s against Matrix Size, illustrating performance improvements and execution times.
Cholesky on the Cell, $Ax=b$, $A=A^T$, $x^T A x > 0$
Cholesky - Using 2 Cell Chips

**SPOTRF - QS20 - 2 CELL BEs**

![Graph showing performance of SPOTRF on QS20 with 2 Cell BEs.

The graph plots Gflop/s against size, with data points for QS20 Peak, SGEMM Peak, and SPOTRF.

The performance peaks at around 400 Gflop/s for large sizes, with a noticeable increase as the size increases.

**Image of a QS20 board with 2 Cell BEs.**
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
  • What about 16 bit floating point?
    • 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 = \(- A\backslash (b - Ax)\)
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

Alfredo Buttari, UTK
Julien Langou, UColorado
Julie Langou, UTK
Piotr Luszczek, MathWorks
Jakub Kurzak, UTK
Stan Tomov, UTK