Scheduling for Numerical Linear Algebra Library at Scale

Jack Dongarra

University of Tennessee
Oak Ridge National Laboratory
University of Manchester
Overview

- Quick look at High Performance Computing
  - Top500
- Design of a the next generation linear algebra library
- 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

11.7 PF/s

10 Pflop/s

IBM Roadrunner

1.02 PF/s

90 T F /

NEC Earth Simulator

100 Tflop/s

IBM BlueGene/L

9.0 TF/s

6-8 years

#1

SUM

1.17 TF/s

Fujitsu 'NWT'

11 Gflop/s

100 Gflop/s

My Laptop

1 Gflop/s

10 Gflop/s

IBM ASCI White

1 T flop/s

100 T flop/s

Intel ASCI Red

1 Tflop/s

10 Tflop/s

59.7 GF/s

100 Gflop/s

100 Mflop/s

0.4 GF/s


#500
Distribution of the Top500 (6/08)

12 systems > 100 Tflop/s
34 systems > 50 Tflop/s
372 systems > 10 Tflop/s

287 systems are used industry.

301 systems were replaced in the June 2008 list.
ASCI Red Compared to ASC Roadrunner

• **ASCI Red Computer**
  - 1997
  - First TFlop/s Computer
  - ~10,000 Pentium Pro
  - ~1 MWatt
  - 200 MHz
  - 200 MFlop/s each proc
    • 104 cabinets
  - ~2500 sq ft (230 m²).

• **ASC Roadrunner**
  - 2008
  - First PFlop/s Computer
  - Hybrid: ~130,000 cores
    • 6,948 dual-core Opterons + 12,960 Cell processors
  - ~3 MWatts
  - 1.8 GHz Opteron + 3.2 GHz Cell
  - ~400 GFlop/s Node
    • Node: 2 socket (each dual core Opteron; each Opteron core has a Cell Chip)
    • 180 nodes/connect unit
    • 18 connect units in the system
  - ~5500 sq ft
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)

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

Hybrid Design (2 kinds of chips & 3 kinds of cores)
Programming required at 3 levels.
# Top10 of the June 2008 List

<table>
<thead>
<tr>
<th>Rank</th>
<th>Computer</th>
<th>Rmax [TF/s]</th>
<th>Rmax / Rpeak</th>
<th>Installation Site</th>
<th>Country</th>
<th>#Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IBM / Roadrunner BladeCenter QS22/LS21</td>
<td>1,026</td>
<td>75%</td>
<td>DOE/NNSA/LANL</td>
<td>USA</td>
<td>122,400</td>
</tr>
<tr>
<td>2</td>
<td>IBM / BlueGene/L eServer Blue Gene Solution</td>
<td>478</td>
<td>80%</td>
<td>DOE/NNSA/LLNL</td>
<td>USA</td>
<td>212,992</td>
</tr>
<tr>
<td>3</td>
<td>IBM / Intrepid Blue Gene/P Solution</td>
<td>450</td>
<td>81%</td>
<td>DOE/OS/ANL</td>
<td>USA</td>
<td>163,840</td>
</tr>
<tr>
<td>4</td>
<td>SUN / Ranger SunBlade x6420</td>
<td>326</td>
<td>65%</td>
<td>NSF/TACC</td>
<td>USA</td>
<td>62,976</td>
</tr>
<tr>
<td>5</td>
<td>CRAY / Jaguar Cray XT4 QuadCore</td>
<td>205</td>
<td>79%</td>
<td>DOE/OS/ORNL</td>
<td>USA</td>
<td>30,976</td>
</tr>
<tr>
<td>6</td>
<td>IBM / JUGENE Blue Gene/P Solution</td>
<td>180</td>
<td>81%</td>
<td>Forschungszentrum Juelich (FZJ)</td>
<td>Germany</td>
<td>65,536</td>
</tr>
<tr>
<td>7</td>
<td>SGI / Encanto SGI Altix ICE 8200</td>
<td>133.2</td>
<td>77%</td>
<td>New Mexico Computing Applications Center</td>
<td>USA</td>
<td>14,336</td>
</tr>
<tr>
<td>8</td>
<td>HP / EKA Cluster Platform 3000 BL460c</td>
<td>132.8</td>
<td>77%</td>
<td>Computational Research Lab, TATA SONS</td>
<td>India</td>
<td>14,384</td>
</tr>
<tr>
<td>9</td>
<td>IBM / Blue Gene/P Solution</td>
<td>112</td>
<td>81%</td>
<td>IDRIS</td>
<td>France</td>
<td>40,960</td>
</tr>
<tr>
<td>10</td>
<td>SGI / Altix ICE 8200EX</td>
<td>106</td>
<td>86%</td>
<td>Total Exploration Production</td>
<td>France</td>
<td>10,240</td>
</tr>
</tbody>
</table>
## Top10 of the June 2008 List

<table>
<thead>
<tr>
<th>Rank</th>
<th>Computer</th>
<th>Rmax [TF/s]</th>
<th>Rmax / Rpeak</th>
<th>Installation Site</th>
<th>Country</th>
<th>#Cores</th>
<th>Power [MW]</th>
<th>MFlops/ Watt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IBM / Roadrunner</td>
<td>1,026</td>
<td>75%</td>
<td>DOE/NNSA/LANL</td>
<td>USA</td>
<td>122,400</td>
<td>2.35</td>
<td>437</td>
</tr>
<tr>
<td></td>
<td>BladeCenter QS22/LS21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>IBM / BlueGene/L eServer Blue Gene Solution</td>
<td>478</td>
<td>80%</td>
<td>DOE/NNSA/LLNL</td>
<td>USA</td>
<td>212,992</td>
<td>2.33</td>
<td>205</td>
</tr>
<tr>
<td>3</td>
<td>IBM / Intrepid Blue Gene/P Solution</td>
<td>450</td>
<td>81%</td>
<td>DOE/OS/ANL</td>
<td>USA</td>
<td>163,840</td>
<td>1.26</td>
<td>357</td>
</tr>
<tr>
<td>4</td>
<td>SUN / Ranger</td>
<td>326</td>
<td>65%</td>
<td>NSF/TACC</td>
<td>USA</td>
<td>62,976</td>
<td>2.00</td>
<td>163</td>
</tr>
<tr>
<td></td>
<td>SunBlade x6420</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>CRAY / Jaguar Cray XT4 QuadCore</td>
<td>205</td>
<td>79%</td>
<td>DOE/OS/ORNL</td>
<td>USA</td>
<td>30,976</td>
<td>1.58</td>
<td>130</td>
</tr>
<tr>
<td>6</td>
<td>IBM / JUGENE Blue Gene/P Solution</td>
<td>180</td>
<td>81%</td>
<td>Forschungszentrum Juelich (FZJ)</td>
<td>Germany</td>
<td>65,536</td>
<td>0.50</td>
<td>357</td>
</tr>
<tr>
<td>7</td>
<td>SGI / Encanto SGI Altix ICE 8200</td>
<td>133.2</td>
<td>77%</td>
<td>New Mexico Computing Applications Center</td>
<td>USA</td>
<td>14,336</td>
<td>0.86</td>
<td>155</td>
</tr>
<tr>
<td>8</td>
<td>HP / EKA Cluster Platform 3000 BL460c</td>
<td>132.8</td>
<td>77%</td>
<td>Computational Research Lab, TATA SONS</td>
<td>India</td>
<td>14,384</td>
<td>0.79</td>
<td>169</td>
</tr>
<tr>
<td>9</td>
<td>IBM / Blue Gene/P Solution</td>
<td>112</td>
<td>81%</td>
<td>IDRIS</td>
<td>France</td>
<td>40,960</td>
<td>0.32</td>
<td>357</td>
</tr>
<tr>
<td>10</td>
<td>SGI / Altix ICE 8200EX</td>
<td>106</td>
<td>86%</td>
<td>Total Exploration Production</td>
<td>France</td>
<td>10,240</td>
<td>0.44</td>
<td>240</td>
</tr>
</tbody>
</table>
Over the next 5 years ORNL/UTK will deploy 2 large Petascale systems

Using 4 MW today, going to 15 MW before year end

By 2012 could be using more than 50 MW!!

Cost estimates based on $0.07 per KwH

Power becomes the architectural driver for future large systems

Cost Per Year
Includes both DOE and NSF systems.
Something’s Happening Here...

From K. Olukotun, L. Hammond, H. Sutter, and B. Smith

• In the “old days” it was:
  - Each year processors would become faster
  - A hardware issue just became a software problem

• Today the clock speed is fixed or getting slower

• Things are still doubling every 18-24 months

• Moore’s Law reinterpreted.
  - Number of cores double every 18-24 months
Today’s Multicores
98% of Top500 Systems Are Based on Multicore

- 282 use Quad-Core
- 204 use Dual-Core
- 3 use Nona-core

- IBM Cell (9 cores)
- Intel Clovertown (4 cores)
- Sun Niagra2 (8 cores)
- SciCortex (6 cores)
- Intel Polaris (80 cores)
- AMD Opteron (4 cores)
- IBM BG/P (4 cores)
And then there’s the GPGPU’s NVIDIA’s Tesla T10P

- **T10P chip**
  - 240 cores; 1.5 GHz
  - Tpeak 1 Tflop/s - 32 bit floating point
  - Tpeak 100 Gflop/s - 64 bit floating point

- **S1070 board**
  - 4 - T10P devices;
  - 700 Watts

- **GTX 280**
  - 1 - T10P; 1.3 GHz
  - Tpeak 864 Gflop/s - 32 bit floating point
  - Tpeak 86.4 Gflop/s - 64 bit floating point
Intel’s Line of Graphics Chips Could Have Broader Uses

By JOHN MARKOFF

SAN FRANCISCO — Intel is planning to release on Monday the first technical details of a new family of chips intended to soup up computer graphics and, eventually, a broad range of computing tasks.

The new microprocessor family, code-named Larrabee, will be available in late 2000 or early 2010. Intel is releasing the details of its plans ahead of the Siggraph industry conference in Los Angeles, which starts Aug. 11.

The company said it would initially aim Larrabee at the personal-computer graphics market, where its “many-core” design, x86 instruction set, which will allow the chips to take advantage of a huge library of existing software.

In 2004, after finding that it could not make its chips faster because they were overheating, Intel adopted a strategy it referred to as a “right-hand turn.” It switched to improving performance by increasing the number of processing elements, or cores, on each chip. That led first to dual-core and now quad-core chips.

Analysts said the first generation of Larrabee may have 16 to 48 cores, depending on the performance goal.

Intel has tried several approaches to chip design, but none of them have had the impact of its x86 family, which was originally introduced three decades ago. Architectures that have been less successful include the Itanium and the 486, neither of which was adopted in mainstream computing.

- Many X 86 IA cores
  - Scalable to Tflop/s
- New cache architecture
- New vector instructions set
  - Vector memory operations
  - Conditionals
  - Integer and floating point arithmetic
- New vector processing unit / wide SIMD
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
ManyCore - Parallelism for the Masses

• We are looking at the following concepts in designing the next numerical library implementation
  ▪ Dynamic Data Driven Execution
  ▪ Self Adapting
  ▪ Block Data Layout
  ▪ Mixed Precision in the Algorithm
  ▪ Exploit Hybrid Architectures
  ▪ Fault Tolerant Methods
# A New Generation of Software:

Software/Algorithms follow hardware evolution in time

<table>
<thead>
<tr>
<th>LINPACK (70’s)</th>
<th>Rely on</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Vector operations)</td>
<td>- Level-1 BLAS operations</td>
</tr>
</tbody>
</table>

Those new algorithms need new kernels and rely on efficient scheduling algorithms.

- Removes a lot of dependencies among the tasks (multicore, distributed computing)
- Avoid latency (distributed computing, out-of-core)
- Rely on fast kernels
# A New Generation of Software:

Software/Algorithms follow hardware evolution in time

<table>
<thead>
<tr>
<th>Software/Algorithms</th>
<th>Follow Hardware Evolution in Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINPACK (70’s)</td>
<td>Rely on Level-1 BLAS operations</td>
</tr>
<tr>
<td>(Vector operations)</td>
<td></td>
</tr>
<tr>
<td>LAPACK (80’s)</td>
<td>Rely on Level-3 BLAS operations</td>
</tr>
<tr>
<td>(Blocking, cache</td>
<td></td>
</tr>
<tr>
<td>friendly)</td>
<td></td>
</tr>
</tbody>
</table>

Those new algorithms need new kernels and rely on efficient scheduling algorithms.

- Removes a lot of dependencies among the tasks (multicore, distributed computing)
- Avoid latency (distributed computing, out-of-core)
- Rely on fast kernels
A New Generation of Software:

<table>
<thead>
<tr>
<th>Software/Algorithms follow hardware evolution in time</th>
<th>LINPACK (70’s) (Vector operations)</th>
<th>Rely on - Level-1 BLAS operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAPACK (80’s) (Blocking, cache friendly)</td>
<td>Rely on - Level-3 BLAS operations</td>
<td></td>
</tr>
<tr>
<td>ScaLAPACK (90’s) (Distributed Memory)</td>
<td>Rely on - PBLAS Mess Passing</td>
<td></td>
</tr>
</tbody>
</table>

Those new algorithms need new kernels and rely on efficient scheduling algorithms.

- removes a lot of dependencies among the tasks, (multicore, distributed computing)
- avoid latency (distributed computing, out-of-core)
- rely on fast kernels
## A New Generation of Software:
### Parallel Linear Algebra Software for Multicore Architectures (PLASMA)

<table>
<thead>
<tr>
<th>Software/Algorithms follow hardware evolution in time</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINPACK (70’s) (Vector operations)</td>
</tr>
<tr>
<td>Rely on - Level-1 BLAS operations</td>
</tr>
<tr>
<td>LAPACK (80’s) (Blocking, cache friendly)</td>
</tr>
<tr>
<td>Rely on - Level-3 BLAS operations</td>
</tr>
<tr>
<td>ScaLAPACK (90’s) (Distributed Memory)</td>
</tr>
<tr>
<td>Rely on - PBLAS Mess Passing</td>
</tr>
<tr>
<td>PLASMA (00’s) New Algorithms (many-core friendly)</td>
</tr>
<tr>
<td>Rely on - a DAG/scheduler - block data layout - some 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.
Coding for an Abstract Multicore

Parallel software for multicore processors should have the following two characteristics:

- **Fine granularity:**
  - High level of parallelism is needed.
  - Cores will probably be associated with relatively small local memories. This requires splitting an operation into tasks that operate on small portions of data in order to reduce bus traffic and improve data locality.

- **Asynchronicity:**
  - As the degree of thread-level parallelism grows and granularity of the operations becomes smaller, the presence of synchronization points in a parallel execution seriously affects the efficiency of an algorithm.
LAPACK and ScaLAPACK

Parallelism:
- LAPACK
- Threaded BLAS
- PThreads
- OpenMP

ScaLAPACK:
- PBLAS
- BLACS
- Mess Passing (MPI, PVM, ...)

About 1 million lines of code
Steps in the LAPACK LU

- **DGETF2** (Factor a panel)
- **DLSWP** (Backward swap)
- **DLSWP** (Forward swap)
- **DTRSM** (Triangular solve)
- **DGEMM** (Matrix multiply)

**LAPACK**

**BLAS**
LU Timing Profile (4 core system)

Threads – no lookahead

Time for each component

Bulk Sync Phases

DGETF2
DLASWP(L)
DLASWP(R)
DTRSM
DGEMM
Adaptive Lookahead - Dynamic

Event Driven Multithreading

Ideas not new.

Many papers use the DAG approach.

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
Achieving Fine Granularity

Fine granularity may require novel data formats to overcome the limitations of BLAS on small chunks of data.

*Column-Major*
Achieving Fine Granularity

Fine granularity may require novel data formats to overcome the limitations of BLAS on small chunks of data.

Column-Major

Blocked
LU – 16 Core (8 socket dual core Opteron)
2.2 GHz

1. LAPACK (BLAS Fork-Join Parallelism)
2. ScaLAPACK (Mess Pass using mem copy)
3. DAG Based (Dynamic Scheduling)
4. Intel MKL Library

Problem Size

GFlop/s
Cholesky on the CELL

- **1 CELL (8 SPEs)**
  - 186 Gflop/s
  - 91 % peak
  - 97 % SGEMM peak

- **2 CELLs (16 SPEs)**
  - 365 Gflop/s
  - 89 % peak
  - 95 % SGEMM peak

Single precision results on the Cell
Simple Cilk Implementation

```c
for (k = 0; k < TILES; k++)
{
    spawn tile_dpotrf(k, MATRIX);
    sync;

    for (m = k+1; m < TILES; m++)
    {
        spawn tile_dtrsm(k, m, MATRIX);
    }
    sync;

    for (m = k+1; m < TILES; m++)
    {
        for (n = k+1; n < m; n++)
        {
            spawn tile_dgemm(n, m, k, MATRIX);
        }
    }
    sync;

    for (m = k+1; m < TILES; m++)
    {
        spawn tile_dsyrk(m, k, MATRIX);
    }
    sync;
}
```

- Writing the driver routine is relatively straightforward after tasks are defined
- Data structures don't need to be redesigned
- LAPACK-style matrices can be used
- However, one needs to pick the algorithm carefully
- Aggressive version needs to be chosen over lazy version
- Right looking Cholesky will deliver some parallelism
- Left looking and top looking will cause the scheduler to serialize the tasks
Barcelona’s SMPSs

- Little understanding of parallelization issues required from the programmer
- Programmer freed from task dependency analysis.
- It is only required that
  - Arguments of task routines are correctly marked as input, output or inout
- Driver routine is put between \#pragma css start and \#pragma css finish
- Complete redesign of data structures may be required
- Impossible to use LAPACK matrices
- The matrix has to be a 2D array of 2D arrays

```c
#pragma css start
for (k = 0; k < TILES; k++)
{
    tile_dpotrf(MATRIX[k][k]);
    for (m = k+1; m < TILES; m++)
    {
        tile_dtrsm(MATRIX[k][k], MATRIX[k][m]);
    }
    for (m = k+1; m < TILES; m++)
    {
        for (n = k+1; n < m; n++)
        {
            tile_dgemm(MATRIX[k][n], MATRIX[k][m],
                        MATRIX[n][m]);
        }
        tile_dsyrk(MATRIX[m][m], MATRIX[k][m]);
    }
} #pragma css finish
```
Parallel Tasks in LU
while (k < TILES && m < TILES) {
    next_n = n; next_m = m; next_k = k;
    next_n++;
    if (next_n > next_k) {
        next_m += cores_num;
        while (next_m >= TILES && next_k < TILES) {
            next_k++;
            next_m = next_m - TILES + next_k;
        }
        next_n = 0;
    }
    if (m == k) {
        if (n == k) {
            TILES_spotrf(k, MATRIX);
            core_progress[TILES*k+k] = 1;
        }
    } else {
        while(core_progress[TILES*k+n] != 1);
        TILES_ssyrk(k, n, MATRIX);
    }
    if (n == k) {
        TILES_strsm(k, m, MATRIX);
        core_progress[TILES*m+k] = 1;
    } else {
        TILES_sgemm(k, m, n, MATRIX);
    }
    n = next_n; m = next_m; k = next_k;
}
Dynamic - Today Hand Coded

- Expert-level understanding of the algorithm required
- Expert-level understanding of parallelism required
- Case-by-case design for each algorithm
- Hundreds of lines of code
  - Complex
  - Error prone
  - Hard to debug
If We Had A Small Matrix Problem

- We would generate the DAG, find the critical path and execute it.
- DAG too large to generate ahead of time
  - Not explicitly generate
  - Dynamically generate the DAG as we go
- Machines will have large number of cores in a distributed fashion
  - Will have to engage in message passing
  - Distributed management
  - Locally have a run time system
The DAGs are Large

• Here is the DAG for the QR factorization on a 20 x 20 matrix
The Nodes or Cores Will Have a Run Time System

**BIN 1**
- Some dependencies satisfied
- Waiting for all dependencies

**BIN 2**
- All dependencies satisfied
- Some data delivered
- Waiting for all data

**BIN 3**
- All data delivered
- Waiting for execution
DAG and Scheduling

- DAG is dynamically generated and implicit, perhaps incrementally evolving
- Everything designed for distributed memory systems

Run time system:
- Unfold more of the DAG
- Bin 1
  - See if new data has arrived
- Bin 2
  - See if new dependences are satisfied
  - If so move task to Bin 3
- Bin 3
  - Exec a task that’s ready
  - Notify children of completion
  - Send data to children
  - If no work do work stealing
Some Questions

• What’s the best way to represent the DAG?
• What’s the best approach to dynamically generating the DAG?
• What run time system should we use?
  ▪ We will probably build something that we would target to the underlying system’s RTS.
• What about work stealing?
  ▪ Can we do better than nearest neighbor work stealing?
• What does the program look like?
  ▪ Experimenting with Cilk, Charm++, UPC, Intel Threads
  ▪ I would like to reuse as much of the existing software as possible
Advanced Architectures
Findings

- Exascale systems are likely feasible by 2017±2
- 10-100 Million processing elements (cores or mini-cores) with chips perhaps as dense as 1,000 cores per socket, clock rates will grow more slowly
- 3D packaging likely
- Large-scale optics based interconnects
- 10-100 PB of aggregate memory
- > 10,000’s of I/O channels to 10-100 Exabytes of secondary storage, disk bandwidth to storage ratios not optimal for HPC use
- Hardware and software based fault management
- Simulation and multiple point designs will be required to advance our understanding of the design space
- Achievable performance per watt will likely be the primary measure of progress

ASCAC Meeting, Washington D.C., August 15, 2007

Achievable performance per watt will likely be the primary measure of progress.
Advanced Architectures

Challenges

- Performance per watt — stretch goal 100 GF/watt of sustained performance ⇒ >> 10 - 100 MW Exascale system
  - Leakage current dominates power consumption
  - Active power switching will help manage standby power
  - Today Roadrunner ~ 2.35 MW; < 500 MF/watt

- Large-scale integration — need to package 10M-100M cores, memory and interconnect
  - 3D packaging likely, goal of small part classes/counts

- Heterogeneous or Homogenous cores?
  - Mini cores or leverage from mass market systems

- Reliability — needs to increase by $10^3$ in faults per PF to achieve MTBF of 1 week
  - Integrated HW/SW management of faults

- Integrated programming models (PGAS?)
  - Provide a usable programming model for hosting existing and future codes
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
Marc Baboulin UCoimbra
Alfredo Buttari, ENS/INRIA
Bilel Hadri, UTK
Julien Langou, UColorado
Julie Langou, UTK
Hatem Ltaief, UTK
Piotr Luszczek, MathWorks
Jakub Kurzak, UTK
Stan Tomov, UTK
PLASMA Collaborators

- **U Tennessee, Knoxville**
  - Jack Dongarra, Julie Langou, Stan Tomov, Jakub Kurzak, Hatem Ltaief, Alfredo Buttari, Julien Langou, Piotr Luszczek, Marc Baboulin, Bilel Hadri, Emmanuel Agullo

- **UC Berkeley**
  - Jim Demmel, Ming Gu, W. Kahan, Beresford Parlett, Xiaoye Li, Osni Marques, Yozo Hida, Jason Riedy, Vasily Volkov, Christo Voemel, David Bindel

- **Other Academic Institutions**
  - UC Davis, CU Denver, Florida IT, Georgia Tech, U Maryland, North Carolina SU, UC Santa Barbara, UT Austin, LBNL
  - TU Berlin, ETH, U Electrocomm. (Japan), FU Hagen, U Carlos III Madrid, U Manchester, U Umeå, U Wuppertal, U Zagreb, UPC Barcelona, ENS Lyon, INRIA, ENSEEIHT

- **Industrial Partners**
  - Cray, HP, Intel, Interactive Supercomputing, MathWorks, NAG, NVIDIA, Microsoft