Communication Avoiding Algorithms in Plasma and Magma

Bilel Hadri, Hatem Ltaief, Emmanuel Agullo, Fengguang Song, and Jack Dongarra

University of Tennessee
Oak Ridge National Laboratory
# Potential System Architecture

<table>
<thead>
<tr>
<th>Systems</th>
<th>2009</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</td>
<td>O(100)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[.03 Bytes/Flop]</td>
<td></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</td>
<td>O(100)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[.002 Bytes/Flop]</td>
<td></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</td>
<td>O(100)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(1:4 or 1:8 from memory BW)</td>
<td></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>
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

Software infrastructure does not exist today
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:
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><strong>LINPACK (70’s)</strong> (Vector operations)</td>
</tr>
<tr>
<td>Rely on</td>
</tr>
<tr>
<td>- Level-1 BLAS operations</td>
</tr>
</tbody>
</table>

| **LAPACK (80’s)** (Blocking, cache friendly)         |
| Rely on                                               |
| - Level-3 BLAS operations                             |

| **ScaLAPACK** (90’s) (Distributed Memory)            |
| Rely on                                               |
| - PBLAS Mess Passing                                  |

| **PLASMA (00’s)** (New Algorithms) (many-core friendly) |
| Rely on                                               |
| - a DAG/scheduler                                     |
| - block data layout                                   |
| - some extra kernels                                   |

| **5IPTFOFXBMHPSJUINT**                                 |
| **IBWFBWFSZ**                                         |
| **MPXHSBOVMBSJUZ**                                    |
| **UIFZTDBMFWFSZXFMM**                                 |
| **NVMUJDPSF**                                         |
| **QFUBTDBMFDPNQVUJOH**                                |

| **SFNPWFTBMPUTPGEFQFOEFODJFT**                        |
| **BNPOHUIFUBTLT**                                     |
| **NVMUJDPSF**                                         |
| **EJTUSJCVUFEDPNQVUJOH**                              |

| **BWPJEMBUFODZ**                                      |
| **EJTUSJCVUFEDPNQVUJOH**                              |
| **PVUPGDPSF**                                         |

| **SFMZPOGBTULFSOFMT**                                 |
| **5IPTFOFXBMHPSJUINTOFFEOFXLFSOFMTBOESFMZPOFGGJDJFOUTDIFEVMJOHBMHPSJUINT** |
## Software/Algorithms follow hardware evolution in time

<table>
<thead>
<tr>
<th>Software/Algorithm</th>
<th>70’s</th>
<th>80’s</th>
<th>90’s</th>
<th>00’s</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINPACK (70’s)</td>
<td>Level-1 BLAS operations</td>
<td>Level-3 BLAS operations</td>
<td>PBLAS Mess Passing</td>
<td></td>
</tr>
<tr>
<td>LAPACK (80’s)</td>
<td>Level-3 BLAS operations</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ScaLAPACK (90’s)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLASMA (00’s)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **LINPACK (70’s)**: Vector operations
- **LAPACK (80’s)**: Blocking, cache friendly
- **ScaLAPACK (90’s)**: Distributed Memory
- **PLASMA (00’s)**: New Algorithms - many-core friendly
### 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>
<th>LINPACK (70’s) (Vector operations)</th>
<th>LAPACK (80’s) (Blocking, cache friendly)</th>
<th>ScaLAPACK (90’s) (Distributed Memory)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rely on</td>
<td>Rely on - Level-1 BLAS operations</td>
<td>Rely on - Level-3 BLAS operations</td>
<td>Rely on - PBLAS Mess Passing</td>
</tr>
<tr>
<td><strong>LINPACK (70’s)</strong></td>
<td>![Image]</td>
<td>![Image]</td>
<td>![Image]</td>
</tr>
<tr>
<td><strong>LAPACK (80’s)</strong></td>
<td>![Image]</td>
<td>![Image]</td>
<td>![Image]</td>
</tr>
<tr>
<td><strong>ScaLAPACK (90’s)</strong></td>
<td>![Image]</td>
<td>![Image]</td>
<td>![Image]</td>
</tr>
</tbody>
</table>
## 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)</td>
</tr>
<tr>
<td>(Vector operations)</td>
</tr>
<tr>
<td>Rely on</td>
</tr>
<tr>
<td>- Level-1 BLAS operations</td>
</tr>
<tr>
<td>LAPACK (80’s)</td>
</tr>
<tr>
<td>(Blocking, cache friendly)</td>
</tr>
<tr>
<td>Rely on</td>
</tr>
<tr>
<td>- Level-3 BLAS operations</td>
</tr>
<tr>
<td>ScaLAPACK (90’s)</td>
</tr>
<tr>
<td>(Distributed Memory)</td>
</tr>
<tr>
<td>Rely on</td>
</tr>
<tr>
<td>- PBLAS Mess Passing</td>
</tr>
<tr>
<td>PLASMA (00’s)</td>
</tr>
<tr>
<td>New Algorithms</td>
</tr>
<tr>
<td>(many-core friendly)</td>
</tr>
<tr>
<td>Rely on</td>
</tr>
<tr>
<td>- a DAG/scheduler</td>
</tr>
<tr>
<td>- block data layout</td>
</tr>
<tr>
<td>- 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.
QR Factorization Intel 16 cores
Tall Skinny Matrices

Theoretical Peak
DGEMM Peak
"MKL (10.1)"
"LAPACK (3.2)"

M=51200 x N
• Fork-join, bulk synchronous processing
Parallel Tasks in QR

- Break into smaller tasks and remove dependencies
Parallel Tasks in QR

Step 1: QR of block 1,1
Parallel Tasks in QR

Step 1: QR of block 1,1

Step 2: Use R to zero $A_{1,2}$
Parallel Tasks in QR

Step 1: QR of block 1,1

Step 2: Use R to zero $A_{1,2}$
Parallel Tasks in QR

Step 1: QR of block $1,1$

Step 2: Use R to zero $A_{1,2}$

Step 3: Use R to zero $A_{1,3}$
Parallel Tasks in QR

Step 1: QR of block 1,1

Step 2: Use R to zero $A_{1,2}$

Step 3: Use R to zero $A_{1,3}$
Parallel Tasks in QR

Step 1: QR of block 1,1

Step 2: Use R to zero $A_{1,2}$

Step 3: Use R to zero $A_{1,3}$
QR Factorization Intel 16 cores
Tall Skinny Matrices

Theoretical Peak
DGEMM Peak
PLASMA (2.1)
"MKL (10.1)"
"LAPACK (3.2)"

Gflop/s

M=51200 x N
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**
  - DAG scheduling
  - explicit parallelism
  - implicit communication
  - Fine granularity / block data layout

- **Arbitrary DAG with dynamic scheduling**

![Diagram of DAG scheduling]

Tile QR factorization

**Tiles**

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># words</td>
<td>( \Theta(mn^2) )</td>
</tr>
<tr>
<td># messages</td>
<td>( \Theta(mn^2/\sqrt{W}) )</td>
</tr>
<tr>
<td># messages</td>
<td>( \Theta(mn^2/W^{3/2}) )</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
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
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
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
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
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
**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
### 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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

*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
Example with 4 and 8 Domains

Execution Trace

Fig. 11. Parallel execution traces of SP-16 with MT=32 and NT=4 on 8 cores.

TABLE III
IMPROVEMENT OF SP-CAQR AGAINST OTHER LIBRARIES (PERFORMANCE RATIO).

<table>
<thead>
<tr>
<th>Matrix sizes</th>
<th>PLASMA</th>
<th>MKL</th>
<th>ScaLAPACK</th>
<th>LAPACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>51200 – 200</td>
<td>9.54</td>
<td>8.77</td>
<td>3.38</td>
<td>28.63</td>
</tr>
<tr>
<td>51200 – 3200</td>
<td>1.27</td>
<td>4.10</td>
<td>2.88</td>
<td>11.05</td>
</tr>
</tbody>
</table>

16 core run
Communication Reducing QR Factorization

(a) One domain: SP-1 (or PLASMA-like Tile QR factorization)
QR Factorization Intel 16 cores
Tall Skinny Matrices

M = 51200 x N

- Theoretical Peak
- DGEMM Peak
- CAQR
- PLASMA (2.1)
- ‘MKL (10.1)’
- ‘LAPACK (3.2)’
Cluster Experiment

- grig.sinrg.cs.utk.edu
- 61 nodes
  - Two CPUs per node
  - Intel Xeon 3.20GHz
  - Peak performance 6.4 GFLOPS
  - Myrinet interconnection (MX 1.0.0)
- Goto BLAS 1.26
  - DGEMM performance 5.57 GFLOPS (87%)
- MPICH-MX
- gcc 64 bits
Weak Scalability of CAQR on the Grig Cluster

- **Peak**
- **Serial-DGEMM x #Cores**
Weak Scalability of CAQR on the Grig Cluster

- Peak
- Serial-DGEMM x #Cores
- ScaLAPACK

Total #GFLOPS vs Number of Cores
Weak Scalability of CAQR on the Grig Cluster

- **Peak**
- **Serial-DGEMM x #Cores**
- **Distri. CAQR**
- **ScaLAPACK**

On 1 CPU, the matrix size is 64x8 tiles

On k CPUs, the matrix size is k*64x8 tiles

Tiles are 200x200 blocks
Weak Scalability (8 columns of tiles)

Scalability of CAQR on the Grig Cluster (8 tiles per row)

- Distri. CAQR
- ScalAPACK

GFLOPS per Core vs. Number of Cores
• Architectural trends are forcing major changes to our algorithms
• Communication avoiding algorithms will be critical for performance.
• PLASMA and MAGMA will make use of CA algorithms.