Distributed Dense Linear Algebra on Heterogeneous Architectures

George Bosilca
bosilca@eecs.utk.edu

Innovative Computing Laboratory
COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF TENNESSEE
Factors that Necessitate to 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

ICL
## Linpack Evolution

Software/Algorithms follow hardware evolution in time

<table>
<thead>
<tr>
<th>LINPACK (70’s)</th>
<th>LAPACK (80’s)</th>
<th>ScaLAPACK (90’s)</th>
<th>PLASMA (00’s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Vector operations)</td>
<td>(Blocking, cache friendly)</td>
<td>(Distributed Memory)</td>
<td>New Algorithms (many-core friendly)</td>
</tr>
<tr>
<td>Rely on Level-1 BLAS operations</td>
<td>Rely on Level-3 BLAS operations</td>
<td>Rely on PBLAS Mess Passing</td>
<td>Rely on a DAG/scheduler, block data layout, some extra kernels</td>
</tr>
</tbody>
</table>
Linpack Evolution

<table>
<thead>
<tr>
<th>Software/Algorithms follow hardware evolution in time</th>
</tr>
</thead>
<tbody>
<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 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.
The algorithmic challenge

- Asynchronicity
  - Avoid fork-join (Bulk sync design)
- Dynamic Scheduling
  - Out of order execution
- Fine Granularity
  - Independent block operations
- Locality of Reference
  - Data storage – Block Data Layout

- Tiled
- Column-major / column-major
- Flat (no indirection)
LAPACK LU

Step 1 → Step 2 → Step 3 → Step 4

» Fork-join, bulk synchronous processing
Exposé the intrinsic parallelism: break into smaller tasks and remove dependencies.
Exposé the intrinsic parallelism

» The three amigos

» Example of QR factorization
  » 4 basic kernels

» LU identical to QR except using different kernels

» Cholesky slightly different due to the matrix symmetry
LU DAG representation
A quite simple 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
What is PLASMA?

• Dense linear algebra software library  
  • Linear systems & least squares (LU, Cholesky, QR/LQ)  
  • Eigenvalues & singular values  
• Multicore processors  
  • Multi-socket multi-core / Shared memory systems  
  • NUMA systems  
• What is next:  
  • GPU acceleration – MAGMA  
  • Distributed Memory – DAGuE / DPLASMA
Current Architectures
» DAGuE: the runtime
   » Deploy a DAG on a heterogeneous distributed environment
   » Architecture aware
      » Minimize data movements (in and out the node)
      » Enforce data locality (cache / NUMA / GPU)
   » Move the data across nodes
» DPLASMA: a algebraic description of a DAG
   » Define the data distribution
   » Describe the algorithm at a high level
      » A powerful description language (?)
Kernels description

**POTRF(k)** (high_priority)
- **RW T** <- type = [tile]
  - -> type = [tile]
- **BODY [core]**
  - **CORE**(potrf, (uplo, NB, T, NB, &INFO))
- **END**

**SYRK(k,n)** (high_priority)
- **R A** <- type = [tile]
- **RW T** <- type = [tile]
  - -> type = [tile]
- **BODY [core]**
  - **CORE**(syrk, (Lower, NoTrans, NB, NB, -1.0, A, NB, 1.0, T, NB))
- **END**

**TRSM(k,n)** (high_priority)
- **R T** <- type = [tile]
- **RW C** <- type = [tile]
  - -> type = [tile]
- **BODY [core]**
  - **CORE**(trsm, (Right, Lower, Trans, NonUnit, NB, NB, 1.0, T, NB, C, NB))
- **END**

**GEMM(k,m,n)**
- **R A** <- type = [tile]
- **R B** <- type = [tile]
- **RW C** <- type = [tile]
  - -> type = [tile]
- **BODY [core]**
  - **CORE**(gemm, (NoTrans, Trans, NB, NB, NB, -1.0, B, NB, A, NB, 1.0, C, NB))
- **END**
- **BODY [cuda]**
  - ...
- **END**
**Algebraic dependencies description**

**POTRF(k)**

\[
\text{k} = [0 .. \text{SIZE}-1] \\
; \ (k / \text{rtileSIZE}) \% \text{GRIDrows} \equiv \text{rowRANK} \\
; \ (k / \text{ctileSIZE}) \% \text{GRIDcols} \equiv \text{colRANK} \\
T \leftarrow (k == 0) \ ? \ A(k, k) : T \ \text{SYRK}(k-1, k) \\
\rightarrow T \ \text{TRSM}(k, k+1..\text{SIZE}-1) \\
\rightarrow A(k, k)
\]

\[
; \ (k >= (\text{SIZE} - \text{PRI_CHANGE})) \ ? \ 10 * (\text{SIZE} - k) \\
* (\text{SIZE} - k) * (\text{SIZE} - k) : 1000000000
\]

**SYRK(k,n)**

\[
\text{k} = [0 .. \text{SIZE}-1] \\
\text{n} = [k+1 .. \text{SIZE}-1] \\
; \ (n / \text{rtileSIZE}) \% \text{GRIDrows} \equiv \text{rowRANK} \\
; \ (n / \text{ctileSIZE}) \% \text{GRIDcols} \equiv \text{colRANK} \\
A \leftarrow C \ \text{TRSM}(k, n) \\
T \leftarrow (k == 0) \ ? \ A(n,n) : T \ \text{SYRK}(k-1, n) \\
\rightarrow (n == k+1) \ ? \ T \ \text{POTRF}(k+1) : T \ \text{SYRK}(k+1,n)
\]

\[
; \ (n >= (\text{SIZE} - \text{PRI_CHANGE})) \ ? \ 10 * (\text{SIZE} - n) \\
* (\text{SIZE} - n) * (\text{SIZE} - n) + 1 : 1000000000
\]

**TRSM(k,n)**

\[
\text{k} = [0 .. \text{SIZE}-1] \\
\text{n} = [k+1 .. \text{SIZE}-1] \\
; \ (n / \text{rtileSIZE}) \% \text{GRIDrows} \equiv \text{rowRANK} \\
; \ (k / \text{ctileSIZE}) \% \text{GRIDcols} \equiv \text{colRANK} \\
T \leftarrow T \ \text{POTRF}(k) \\
C \leftarrow (k == 0) \ ? \ A(n, k) : C \ \text{GEMM}(k-1, n, k) \\
\rightarrow A \ \text{SYRK}(k, n) \\
\rightarrow A \ \text{GEMM}(k, n+1..\text{SIZE}-1, n) \\
\rightarrow B \ \text{GEMM}(k, n, k+1..n-1) \\
\rightarrow A(n, k)
\]

\[
; \ (n >= (\text{SIZE} - \text{PRI_CHANGE})) \ ? \ 10 * (\text{SIZE} - n) \\
* (\text{SIZE} - n) * (\text{SIZE} - n) + 2 : 1000000000
\]

**GEMM(k,m,n) (assoc(k))**

\[
\text{k} = [0 .. \text{SIZE}] \\
\text{m} = [k+2 .. \text{SIZE}-1] \\
\text{n} = [k+1 .. m-1] \\
; \ (m / \text{rtileSIZE}) \% \text{GRIDrows} \equiv \text{rowRANK} \\
; \ (n / \text{ctileSIZE}) \% \text{GRIDcols} \equiv \text{colRANK} \\
A \leftarrow C \ \text{TRSM}(k, n) \\
B \leftarrow C \ \text{TRSM}(k, m) \\
C \leftarrow (k == 0) \ ? \ A(m, n) : C \ \text{GEMM}(k-1, m, n) \\
\rightarrow (n == k+1) \ ? \ C \ \text{TRSM}(k+1, m) : \\
\quad C \ \text{GEMM}(k+1, m, n)
\]

\[
; \ (m >= (\text{SIZE} - \text{PRI_CHANGE})) \ ? \ 10 * (\text{SIZE} - m) \\
* (\text{SIZE} - m) * (\text{SIZE} - m) + 1 : 1000000000
\]
Cholesky (4x4)
Heterogeneous multi-GPU
Cholesky (problem size)

Griffon: 81 nodes, 648 cores, Infiniband 20Gbs
Cholesky (weak scalability)
Cholesky (strong scalability)

Theoretical peak
GEMM peak
DPLASMA (NB=340)
DSBP (NB=340)
ScaLAPACK (NB=120)
DPLASMA
Composability

» Totally remove the need for synchronization
» Follow the data flow between algebraic description of DAGs

» Shorten the time to completion
Composability