Communication avoiding algorithms in linear algebra

Laura Grigori

Alpines

INRIA Paris - LJLL, UPMC

November 2016
Plan

• Motivation
• Selected past work on reducing communication
• Communication complexity of linear algebra operations
• Communication avoiding for dense linear algebra
  • LU, QR, Rank Revealing QR factorizations
  • Progressively implemented in ScaLAPACK, LAPACK
  • Algorithms for multicore processors
• Communication avoiding for sparse linear algebra
  • Sparse low rank matrix approximation
  • Krylov subspace methods
• Conclusions
Numerical simulations require increasingly computing power as data sets grow exponentially.

Figures from astrophysics:
- Produce and analyze multi-frequency 2D images of the universe when it was 5% of its current age.
- COBE (1989) collected 10 gigabytes of data, required 1 Teraflop per image analysis.
- PLANCK (2010) produced 1 terabyte of data, requires 100 Petaflops per image analysis.
- Future experiment (2020) estimated to collect .5 petabytes, require 100 Exaflops per image analysis.

Source: J. Borrill, LBNL, R. Stompor, Paris 7

Source: T. Guignon, IFPEN
http://www.epm.ornl.gov/chammp/chammp.html

http://www.scidacreview.org/0704/html/cmb.html
Motivation - the communication wall

• Runtime of an algorithm is the sum of:
  • \#flops x \texttt{time\_per\_flop}
  • \#words\_moved / \texttt{bandwidth}
  • \#messages x \texttt{latency}

• Time to move data >> time per flop
  • Gap steadily and exponentially growing over time
Motivation - the communication wall

• Runtime of an algorithm is the sum of:
  • #flops x time_per_flop
  • #wordsMoved / bandwidth
  • #messages x latency

• Time to move data >> time per flop
  • Gap steadily and exponentially growing over time

<table>
<thead>
<tr>
<th></th>
<th>Annual improvements</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time/flop</td>
</tr>
<tr>
<td>Network</td>
<td>59%</td>
</tr>
<tr>
<td>DRAM</td>
<td>59%</td>
</tr>
</tbody>
</table>

• Performance of an application is less than 10% of the peak performance

“We are going to hit the memory wall, unless something basic changes” [W. Wulf, S. McKee, 95]
Compelling numbers (1)

DRAM bandwidth:
- Mid 90’s ~ 0.2 bytes/flop – 1 byte/flop
- Past few years ~ 0.02 to 0.05 bytes/flop

DRAM latency:
- DDR2 (2007) ~ 120 ns 1x
- DDR4 (2014) ~ 45 ns 2.6x in 7 years
- Stacked memory ~ similar to DDR4 13% / year

Time/flop
- 2006 Intel Yonah ~ 2GHz x 2 cores (32 GFlops/chip) 1x
- 2015 Intel Haswell ~2.3GHz x 16 cores (588 GFlops/chip) 18x in 9 years 34% / year

Source: J. Shalf, LBNL
Compelling numbers (2)

Fetch from DRAM 1 byte of data
• 1988: compute 6 flops
• 2004: compute a few 100 flops
• 2015: compute 26460 flops/chip (see below)

Receive from another proc 1 byte of data:
• Compute 147000 - 1065000 flops

Example of one supercomputer today:
• Intel Haswell: 8 flops per cycle per core
• Interconnect: 0.25 µs to 3.7 µs MPI latency, 8GB/sec MPI bandwidth
The role of numerical linear algebra

• Challenging applications often rely on solving linear algebra problems
• Linear systems of equations
  
  Solve \( Ax = b \), where \( A \in \mathbb{R}^{n \times n} \), \( b \in \mathbb{R}^{n} \), \( x \in \mathbb{R}^{n} \)

  • Direct methods
    
    \[ PA = LU \], then solve \( P^T L U x = b \)

    LU factorization is backward stable,

    \[ \left\| PA - \hat{L} \cdot \hat{U} \right\|_\infty \] is small, close to machine epsilon in practice

  • Iterative methods
    
    • Find a solution \( x_k \) from \( x_0 + K_k (A, r_0) \), where \( K_k (A, r_0) = \text{span} \{ r_0, A r_0, \ldots, A^{k-1} r_0 \} \)
      such that the Petrov-Galerkin condition \( b - Ax_k \perp L_k \) is satisfied,
      
      where \( L_k \) is a subspace of dimension \( k \) and \( r_0 = Ax_0 - b \).

    • Convergence depends on \( \kappa(A) \) and the eigenvalue distribution (for SPD matrices).
Least Square (LS) Problems

• Given $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$, solve $\min_x \|Ax - b\|_2$.

• Any solution of the LS problem satisfies the normal equations: $A^T Ax = A^T b$

• Given the QR factorization of $A$

$$A = Q \begin{bmatrix} R \\ 0 \end{bmatrix}$$

where $R$ is $n \times n$ upper triangular matrix

$Q$ is $m \times m$ orthogonal matrix

if $\text{rank}(A) = \text{rank}(R) = n$, then the LS solution is given by $Rx = (Q^T b)(1 : n)$

• The QR factorization is column-wise backward stable

$$\|A - \hat{Q}\hat{R}\|_2$$ is small, close to machine epsilon in practice
Rank revealing factorizations

• A rank revealing QR (RRQR) factorization is given as

\[ A\Pi = QR = Q \begin{pmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{pmatrix}, \]  

\( R_{11} \) is \( k \)-by-\( k \)

with \( \sigma_{\text{min}}(R_{11}) \geq \frac{\sigma_k(A)}{p(k,n)} \), \( \sigma_{\text{max}}(R_{22}) \leq \sigma_{k+1}(A) \cdot p(k,n) \),

\( p(k,n) \) is a low degree polynomial in \( n \) and \( k \), \( R_{11} \) is well conditioned, \( \|R_{22}\|_2 \) is small

strong RRQR if \( |R_{11}^{-1}R_{12}| \leq c \)

• Since \( \sigma_{k+1}(A) \leq \sigma_{\text{max}}(R_{22}) = \|R_{22}\|_2 \), the numerical rank of \( A \) is \( k \)
• \( Q(:,1:k) \) forms an approximate orthogonal basis for the range of \( A \)

• \( A\Pi \begin{pmatrix} R_{11}^{-1}R_{12} \\ -I \end{pmatrix} = Q \begin{pmatrix} 0 \\ -R_{22} \end{pmatrix} \), then \( \Pi \begin{pmatrix} R_{11}^{-1}R_{12} \\ -I \end{pmatrix} \) are approximate null vectors

• Applications: subset selection and linear dependency analysis, rank determination, low rank approximation - solve \( \min_{\text{rank}(X)=k} \|A-X\| \)
Approaches for reducing communication

• **Tuning**
  - Overlap communication and computation, at most a factor of 2 speedup

• **Same numerical algorithm, different schedule of the computation**
  - Block algorithms for NLA
    - Barron and Swinnerton-Dyer, 1960
    - ScaLAPACK, Blackford et al 97
  - Cache oblivious algorithms for NLA
    - Gustavson 97, Toledo 97, Frens and Wise 03, Ahmed and Pingali 00

• **Same algebraic framework, different numerical algorithm**
  - The approach used in CA algorithms
  - More opportunities for reducing communication, may affect stability
Communication in CMB data analysis

- **Map-making problem**
  - Find the best map $x$ from observations $d$, scanning strategy $A$, and noise $N^{-1}$
  - Solve generalized least squares problem involving sparse matrices of size $10^{12}$-by-$10^7$

- **Spherical harmonic transform (SHT)**
  - Synthesize a sky image from its harmonic representation
    - Computation over rows of a 2D object (summation of spherical harmonics)
    - Communication to transpose the 2D object
    - Computation over columns of the 2D object (FFTs)

---

**Map making**, with R. Stompor, M. Szydlarski
Results obtained on Hopper, Cray XE6, NERSC

**SHT**, with R. Stompor, M. Szydlarski
Simulation on a petascale computer
Evolution of numerical libraries

**LINPACK (70’s)**
- vector operations, uses BLAS1/2
- HPL benchmark based on Linpack LU factorization

**LAPACK (80’s)**
- Block versions of the algorithms used in LINPACK
- Uses BLAS3

**ScaLAPACK (90’s)**
- Targets distributed memories
- 2D block cyclic distribution of data
- PBLAS based on message passing

**PLASMA (2008): new algorithms**
- Targets many-core
- Block data layout
- Low granularity, high asynchronicity

Project developed by U Tennessee Knoxville, UC Berkeley, other collaborators.
Source: inspired from J. Dongarra, UTK, J. Langou, CU Denver
Evolution of numerical libraries

- Did we need new algorithms?
  - Results on two-socket, quad-core Intel Xeon EMT64 machine, 2.4 GHz per core, peak performance 76.5 Gflops/s
  - LU factorization of an m-by-n matrix, m=10^5 and n varies from 10 to 1000

![Graph showing performance comparison of different LU factorization methods.](image)
Motivation

• The communication problem needs to be taken into account higher in the computing stack

• A paradigm shift in the way the numerical algorithms are devised is required

• Communication avoiding algorithms - a novel perspective for numerical linear algebra
  • Minimize volume of communication
  • Minimize number of messages
  • Minimize over multiple levels of memory/parallelism
  • Allow redundant computations (preferably as a low order term)
Communication Complexity of Dense Linear Algebra

• Matrix multiply, using $2n^3$ flops (sequential or parallel)
  • Hong-Kung (1981), Irony/Tishkin/Toledo (2004)
  • Lower bound on Bandwidth = $\Omega \left( \frac{\text{#flops}}{M^{1/2}} \right)$
  • Lower bound on Latency = $\Omega \left( \frac{\text{#flops}}{M^{3/2}} \right)$

• Same lower bounds apply to LU using reduction
  • Demmel, LG, Hoemmen, Langou 2008

\[
\begin{pmatrix}
I & -B \\
A & I \\
I & I
\end{pmatrix} = \begin{pmatrix}
I & I \\
A & I \\
I & I
\end{pmatrix} \begin{pmatrix}
I & -B \\
I & AB \\
I & I
\end{pmatrix}
\]

• And to almost all direct linear algebra [Ballard, Demmel, Holtz, Schwartz, 09]
Lower bounds for linear algebra

- Computation modelled as an n-by-n-by-n set of lattice points
  \((i,j,k)\) represents the operation \(c(i,j) += f_{ij}(g_{ijk}(a(i,k)*b(k,j)))\)
  
  - The computation is divided in S phases
  - Each phase contains exactly M (the fast memory size) load and store instructions
  - Determine how many flops the algorithm can compute in each phase, by applying discrete Loomis-Whitney inequality:

\[
\varepsilon^2 \leq N_A N_B N_C
\]

Algorithms in direct linear algebra:

\[
\begin{align*}
&\text{for } i,j,k = 1: n \\
&\quad c(i,j) = f_{ij}(g_{ijk}(a(i,k),b(k,j))) \\
&\text{endfor}
\end{align*}
\]

- set of points in \(\mathbb{R}^3\), represent \(w\) arithmetics

- orthogonal projections of the points onto coordinate planes \(N_A, N_B, N_C\) represent values of A, B, C
Lower bounds for matrix multiplication (contd)

• Discrete Loomis-Whitney inequality:
  \[ w^2 \leq N_A N_B N_C \]

• Since there are at most 2M elements of A, B, C in a phase, the bound is:
  \[ w \leq 2\sqrt{2} M^{3/2} \]

• The number of phases S is \#flops/w, and hence the lower bound on communication is:

  \[
  \# \text{messages}(S) \geq \frac{\# \text{flops}}{w} = \Omega\left(\frac{\# \text{flops}}{M^{3/2}}\right)
  \]

  \[
  \# \text{loads/stores} \geq \Omega\left(\frac{\# \text{flops}}{M^{1/2}}\right)
  \]
### Sequential algorithms and communication bounds

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Minimizing #words (not #messages)</th>
<th>Minimizing #words and #messages</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cholesky</strong></td>
<td>LAPACK</td>
<td><strong>[Gustavson, 97]</strong>&lt;br&gt;<strong>[Ahmed, Pingali, 00]</strong></td>
</tr>
<tr>
<td><strong>LU</strong></td>
<td>LAPACK (few cases)&lt;br&gt;[Toledo,97], [Gustavson, 97]&lt;br&gt;both use partial pivoting</td>
<td><strong>[LG, Demmel, Xiang, 08]</strong>&lt;br&gt;<strong>[Khabou, Demmel, LG, Gu, 12]</strong>&lt;br&gt;uses tournament pivoting</td>
</tr>
<tr>
<td><strong>QR</strong></td>
<td>LAPACK (few cases)&lt;br&gt;[Elmroth,Gustavson,98]</td>
<td><strong>[Frens, Wise, 03]</strong>, 3x flops&lt;br&gt;[Demmel, LG, Hoemmen, Langou, 08]&lt;br&gt;[Ballard et al, 14]</td>
</tr>
<tr>
<td><strong>RRQR</strong></td>
<td></td>
<td><strong>[Demmel, LG, Gu, Xiang 11]</strong>&lt;br&gt;uses tournament pivoting, 3x flops</td>
</tr>
</tbody>
</table>

- Only several references shown for block algorithms (LAPACK), cache-oblivious algorithms and communication avoiding algorithms
- CA algorithms exist also for SVD and eigenvalue computation
## 2D Parallel algorithms and communication bounds

- If memory per processor = $n^2 / P$, the lower bounds become
  
  \[
  \#\text{words}_\text{moved} \geq \Omega \left( \frac{n^2}{P^{1/2}} \right), \quad \#\text{messages} \geq \Omega \left( P^{1/2} \right)
  \]

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Minimizing #words (not #messages)</th>
<th>Minimizing #words and #messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cholesky</td>
<td>ScaLAPACK</td>
<td>ScaLAPACK</td>
</tr>
<tr>
<td>LU</td>
<td>ScaLAPACK</td>
<td>[LG, Demmel, Xiang, 08]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[Khabou, Demmel, LG, Gu, 12]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uses tournament pivoting</td>
</tr>
<tr>
<td>QR</td>
<td>ScaLAPACK</td>
<td>[Demmel, LG, Hoemmen, Langou, 08]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[Ballard et al, 14]</td>
</tr>
<tr>
<td>RRQR</td>
<td>ScaLAPACK</td>
<td>[Demmel, LG, Gu, Xiang 13]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>uses tournament pivoting, 3x flops</td>
</tr>
</tbody>
</table>

- Only several references shown, block algorithms (ScaLAPACK) and communication avoiding algorithms
- CA algorithms exist also for SVD and eigenvalue computation
The algebra of LU factorization

- Compute the factorization PA = LU

- Given the matrix

\[
A = \begin{pmatrix}
3 & 1 & 3 \\
6 & 7 & 3 \\
9 & 12 & 3
\end{pmatrix}
\]

Let

\[
M_1A = \begin{pmatrix}
1 \\
-2 & 1 \\
-3 & 1
\end{pmatrix},
\quad
M_1A = \begin{pmatrix}
3 & 1 & 3 \\
0 & 5 & -3 \\
0 & 9 & -6
\end{pmatrix}
\]
The need for pivoting

• For stability avoid division by small elements, otherwise $||A-LU||$ can be large
  • Because of roundoff error
  • For example

\[
A = \begin{pmatrix}
0 & 3 & 3 \\
3 & 1 & 3 \\
6 & 2 & 3
\end{pmatrix}
\]

has an LU factorization if we permute the rows of $A$

\[
PA = \begin{pmatrix}
6 & 2 & 3 \\
0 & 3 & 3 \\
3 & 1 & 3
\end{pmatrix} = \begin{pmatrix} 1 & \phantom{0} & \phantom{0} \\
1 & \phantom{0} & \phantom{0} \\
0.5 & 1 & \phantom{0}
\end{pmatrix} \begin{pmatrix}
6 & 2 & 3 \\
0 & 3 & 3 \\
0 & 0 & 1.5
\end{pmatrix}
\]

• Partial pivoting allows to bound all elements of $L$ by 1.
LU with partial pivoting – BLAS 2 algorithm

for i = 1 to n-1
    Let A(j,i) be elt. of max magnitude in A(i+1:n,i)
    Permute rows i and j
    for j = i+1 to n
        A(j,i) = A(j,i)/A(i,i)
    for j = i+1 to n
        for k = i+1 to n
            A(j,k) = A(j,k) - A(j,i) * A(i,k)

- Algorithm using BLAS 1/2 operations

for i = 1 to n-1
    Let A(j,i) be elt. of max magnitude in A(i+1:n,i)
    Permute rows i and j
    A(i+1:n,i) = A(i+1:n,i) * ( 1 / A(i,i) )
    ... BLAS 1 (scale a vector)
    A(i+1:n,i+1:n) = A(i+1:n , i+1:n )
    - A(i+1:n , i) * A(i , i+1:n)
    ... BLAS 2 (rank-1 update)

Source slide: J. Demmel
Block LU factorization – obtained by delaying updates

- Matrix A of size \( nxn \) is partitioned as
  \[
  A = \begin{bmatrix}
  A_{11} & A_{12} \\
  A_{21} & A_{22}
  \end{bmatrix}, \text{ where } A_{11} \text{ is } b \times b
  \]

- The first step computes LU with partial pivoting of the first block:
  \[
  P_1 \begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix} = \begin{bmatrix} L_{11} \\ L_{21} \end{bmatrix} U_{11}
  \]

- The factorization obtained is:
  \[
  P_1 A = \begin{bmatrix} L_{11} \\ L_{21} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ I_{n-b} & A_{22}^1 \end{bmatrix}, \text{ where } U_{12} = L_{11}^{-1} A_{12}
  \]
  \[
  A_{22}^1 = A_{22} - U_{12}
  \]

- The algorithm continues recursively on the trailing matrix \( A_{22}^1 \)
Block LU factorization – the algorithm

1. Compute LU with partial pivoting of the first panel

\[ P_1 \begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix} = \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} U_{11} \]

2. Pivot by applying the permutation matrix \( P_1 \) on the entire matrix

\[ P_1 A = \overline{A} \]

3. Solve the triangular system to compute a block row of \( U \)

\[ U_{12} = L_{12}^{-1} \overline{A}_{12} \]

4. Update the trailing matrix

\[ \overline{A}_{22}^1 = \overline{A}_{22} - L_{21} U_{12} \]

5. The algorithm continues recursively on the trailing matrix \( \overline{A}_{22}^1 \)
LU factorization (as in ScaLAPACK pdgetrf)

LU factorization on a $P = P_r \times P_c$ grid of processors

For $ib = 1$ to $n-1$ step $b$

$A^{(ib)} = A(ib:n, ib:n)$

(1) Compute panel factorization $O(n \log_2 P_r)$
   - find pivot in each column, swap rows

(2) Apply all row permutations $O(n / b (\log_2 P_c + \log_2 P_r))$
   - broadcast pivot information along the rows
   - swap rows at left and right

(3) Compute block row of U $O(n / b \log_2 P_c)$
   - broadcast right diagonal block of L of current panel

(4) Update trailing matrix $O(n / b (\log_2 P_c + \log_2 P_r))$
   - broadcast right block column of L
   - broadcast down block row of U
General scheme for QR factorization by Householder transformations

The Householder matrix

\[ H_i = I - \tau_i h_i h_i^T \]

has the following properties:

- is symmetric and orthogonal,
  \[ H_i^2 = I, \]
- is independent of the scaling of \( h_i \),
- it reflects \( x \) about the hyperplane \( \text{span}(h_i)^\perp \)

- For QR, we choose a Householder matrix that allows to annihilate the elements of a vector \( x \), except first element.
General scheme for
QR factorization by Householder transformations

- Apply Householder transformations to annihilate subdiagonal entries

\[ A = \begin{pmatrix} x & x & x & x \\ x & x & x & x \\ x & x & x & x \\ x & x & x & x \end{pmatrix} = H_1 \begin{pmatrix} x & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \end{pmatrix} = H_1 \begin{pmatrix} 1 & \tilde{H}_2 \\ 0 & 0 & x & x \\ 0 & 0 & x & x \end{pmatrix} = H_1 H_2 \begin{pmatrix} 1 & \tilde{H}_3 \\ 0 & 0 & x & x \\ 0 & 0 & 0 & x \end{pmatrix} = H_1 H_2 H_3 R = QR \]

- For \( A \) of size \( mxn \), the factorization can be written as:

\[
H_n H_{n-1} \ldots H_2 H_1 A = R \rightarrow A = (H_n H_{n-1} \ldots H_2 H_1)^T R \\
Q = H_1 H_2 \ldots H_n
\]
Compact representation for $Q$

- Orthogonal factor $Q$ can be represented implicitly as

$$Q = H_1 H_2 \ldots H_b = (I - \tau_1 h_1 h_1^T) \ldots (I - \tau_b h_b h_b^T) = I - YTY^T,$$

where

$$Y = \begin{pmatrix} h_1 & h_2 & \ldots & h_b \end{pmatrix}$$

- Example for $b=2$:

$$Y = (h_1 | h_2), \quad T = \begin{pmatrix} \tau_1 & -\tau_1 h_1^T h_2 \tau_2 \\ \tau_2 & \end{pmatrix}$$
Algebra of block QR factorization

Matrix $A$ of size $n \times n$ is partitioned as

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \text{ where } A_{11} \text{ is } b \times b$$

Block QR algebra

The first step of the block QR factorization algorithm computes:

$$Q_1^T A = \begin{bmatrix} R_{11} & R_{12} \\ \phantom{R_{11}} & \phantom{R_{12}} \end{bmatrix}$$

The algorithm continues recursively on the trailing matrix $A_{22}^{1}$
Block QR factorization

\[ A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = Q_1 \begin{pmatrix} R_{11} & R_{12} \\ \end{pmatrix} \]

Block QR algebra:

1. Compute panel factorization:
\[ \begin{pmatrix} A_{11} \\ A_{12} \end{pmatrix} = Q_1 \begin{pmatrix} R_{11} \end{pmatrix}, \quad Q_1 = H_1 H_2 \ldots H_b \]

2. Compute the compact representation:
\[ Q_1 = I - Y_1^T Y_1 \]

3. Update the trailing matrix:
\[ \begin{pmatrix} I - Y_1^T Y_1^T \end{pmatrix} \begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix} = \begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix} - Y_1^T \begin{pmatrix} T_1^T \end{pmatrix} \begin{pmatrix} Y_1^T \end{pmatrix} \begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix} = \begin{pmatrix} R_{12} \\ \end{pmatrix} \]

4. The algorithm continues recursively on the trailing matrix.
TSQR: QR factorization of a tall skinny matrix using Householder transformations

• QR decomposition of $m \times b$ matrix $W$, $m \gg b$
  • $P$ processors, block row layout

• Classic Parallel Algorithm
  • Compute Householder vector for each column
  • Number of messages $\propto b \log P$

• Communication Avoiding Algorithm
  • Reduction operation, with QR as operator
  • Number of messages $\propto \log P$

\[ W = \begin{bmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \end{bmatrix} \rightarrow \begin{bmatrix} R_{00} \\ R_{10} \\ R_{20} \\ R_{30} \end{bmatrix} \rightarrow R_{01} \rightarrow R_{11} \rightarrow R_{02} \]

J. Demmel, LG, M. Hoemmen, J. Langou, 08
Parallel TSQR

References: Golub, Plemmons, Sameh 88, Pothen, Raghavan, 89, Da Cunha, Becker, Patterson, 02
Algebra of TSQR

Parallel:

\[
W = \begin{bmatrix}
W_0 \\
W_1 \\
W_2 \\
W_3 
\end{bmatrix} \rightarrow \begin{bmatrix}
R_{00} \\
R_{10} \\
R_{20} \\
R_{30} 
\end{bmatrix} \rightarrow \begin{bmatrix}
R_{01} \\
R_{11} \\
\end{bmatrix} \rightarrow R_{02}
\]

\[
W = \begin{pmatrix}
W_0 \\
W_1 \\
W_2 \\
W_3
\end{pmatrix} = \begin{pmatrix}
Q_{00}R_{00} \\
Q_{10}R_{10} \\
Q_{20}R_{20} \\
Q_{30}R_{30}
\end{pmatrix} = \begin{pmatrix}
Q_{00} \\
Q_{10} \\
Q_{20} \\
Q_{30}
\end{pmatrix} \begin{pmatrix}
R_{00} \\
R_{10} \\
R_{20} \\
R_{30}
\end{pmatrix}
\]

\[
\begin{pmatrix}
R_{00} \\
R_{10} \\
R_{20} \\
R_{30}
\end{pmatrix} = \begin{pmatrix}
Q_{01}R_{01} \\
Q_{11}R_{11}
\end{pmatrix} = \begin{pmatrix}
Q_{01} \\
Q_{11}
\end{pmatrix} \begin{pmatrix}
R_{01} \\
R_{11}
\end{pmatrix} = Q_{02}R_{02}
\]

Q is represented implicitly as a product

Output: \{Q_{00}, Q_{10}, Q_{00}, Q_{20}, Q_{30}, Q_{01}, Q_{11}, Q_{02}, R_{02}\}
Flexibility of TSQR and CAQR algorithms

Parallel: \[ w = \begin{bmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \end{bmatrix} \rightarrow R_{00} \rightarrow R_{01} \rightarrow R_{02} \]

Sequential: \[ w = \begin{bmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \end{bmatrix} \rightarrow R_{00} \rightarrow R_{01} \rightarrow R_{02} \rightarrow R_{03} \]

Dual Core: \[ w = \begin{bmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \end{bmatrix} \rightarrow R_{00} \rightarrow R_{01} \rightarrow R_{02} \rightarrow R_{03} \]

Reduction tree will depend on the underlying architecture, could be chosen dynamically
Algebra of TSQR

Parallel: 

\[ W = \begin{bmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \end{bmatrix} \rightarrow \begin{bmatrix} R_{00} \\ R_{10} \\ R_{20} \\ R_{30} \end{bmatrix} \rightarrow \begin{bmatrix} R_{01} \\ R_{11} \end{bmatrix} \rightarrow R_{02} \]
QR for General Matrices

• Cost of **CAQR** vs **ScalAPACK's PDGEQRF**
  • \( n \times n \) matrix on \( P^{1/2} \times P^{1/2} \) processor grid, block size \( b \)
  • Flops: \( (4/3)n^3/P + (3/4)n^2b \log P/P^{1/2} \) vs \( (4/3)n^3/P \)
  • Bandwidth: \( (3/4)n^2 \log P/P^{1/2} \) vs same
  • Latency: \( 2.5 n \log P / b \) vs \( 1.5 n \log P \)

• Close to optimal (modulo \( \log P \) factors)
  • Assume: \( O(n^2/P) \) memory/processor, \( O(n^3) \) algorithm,
  • Choose \( b \) near \( n / P^{1/2} \) (its upper bound)
  • Bandwidth lower bound:
    \( \Omega(n^2 /P^{1/2}) \) – just \( \log(P) \) smaller
  • Latency lower bound:
    \( \Omega(P^{1/2}) \) – just polylog\((P)\) smaller
Performance of TSQR vs Sca/LAPACK

- Parallel
  - Intel Xeon (two socket, quad core machine), 2010
    - Up to **5.3x speedup** (8 cores, $10^5 \times 200$)
  - Pentium III cluster, Dolphin Interconnect, MPICH, 2008
    - Up to **6.7x speedup** (16 procs, 100K x 200)
  - BlueGene/L, 2008
    - Up to **4x speedup** (32 procs, 1M x 50)
  - Tesla C 2050 / Fermi (Anderson et al)
    - Up to **13x** (110,592 x 100)
  - Grid – **4x** on 4 cities vs 1 city (Dongarra, Langou et al)
  - QR computed locally using recursive algorithm (Elmroth-Gustavson) – enabled by TSQR

- Results from many papers, for some see [Demmel, LG, Hoemmen, Langou, SISC 12], [Donfack, LG, IPDPS 10].
Modeled Speedups of CAQR vs ScaLAPACK

Petascale up to 22.9x
IBM Power 5 up to 9.7x
“Grid” up to 11x

Petascale machine with 8192 procs, each at 500 GFlops/s, a bandwidth of 4 GB/s.

\[ \gamma = 2 \cdot 10^{-12} s, \alpha = 10^{-5} s, \beta = 2 \cdot 10^{-9} s / \text{word}. \]
Impact

• TSQR/CAQR implemented in
  • Intel MKL library
  • GNU Scientific Library
  • ScaLAPACK
  • Spark for data mining

• CALU implemented in
  • Cray’s libsci
  • To be implemented in lapack/scapalack
Algebra of TSQR

Parallel:

\[
W = \begin{bmatrix}
W_0 \\
W_1 \\
W_2 \\
W_3
\end{bmatrix} \rightarrow 
\begin{bmatrix}
R_{00} \\
R_{10} \\
R_{20} \\
R_{30}
\end{bmatrix} \rightarrow 
\begin{bmatrix}
R_{01} \\
R_{11} \\
R_{02}
\end{bmatrix}
\]

TSQR-HR

CAQR

Step 0

Step 1

Step 2
Reconstruct Householder vectors from TSQR

The QR factorization using Householder vectors

\[ W = QR = (I - YTY_1^T)R \]

can be re-written as an LU factorization

\[ W - R = Y(-TY_1^T)R \]
\[ Q - I = Y(-TY_1^T) \]
Reconstruct Householder vectors TSQR-HR

1. Perform TSQR
2. Form Q explicitly (tall-skinny orthonormal factor)
3. Perform LU decomposition: $Q - I = LU$
4. Set $Y = L$
5. Set $T = -U Y_1^{-T}$

$$I - YTY^T = I - \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} T \begin{bmatrix} Y_1^T & Y_2^T \end{bmatrix}$$
Strong scaling

- Hopper: Cray XE6 (NERSC) – 2 x 12-core AMD Magny-Cours (2.1 GHz)
- Edison: Cray CX30 (NERSC) – 2 x 12-core Intel Ivy Bridge (2.4 GHz)
- Effective flop rate, computed by dividing $2m n^2 - 2n^3/3$ by measured runtime

Ballard, Demmel, LG, Jacquelin, Knight, Nguyen, and Solomonik, 2015.
Weak scaling QR on Hopper

- Matrix of size 15K-by-15K to 131K-by-131K
- Hopper: Cray XE6 supercomputer (NERSC) – dual socket 12-core Magny-Cours Opteron (2.1 GHz)
The LU factorization of a tall skinny matrix

First try the obvious generalization of TSQR.

\[
W = \begin{pmatrix}
W_0 \\
W_1 \\
W_2 \\
W_3
\end{pmatrix} = \begin{pmatrix}
\Pi_{00} & \Pi_{10} \\
& \Pi_{20} \\
& & \Pi_{30}
\end{pmatrix} \begin{pmatrix}
L_{00} \\
& L_{10} \\
& & \begin{pmatrix}
L_{20} \\
& L_{30}
\end{pmatrix}
\end{pmatrix} \begin{pmatrix}
U_{00} \\
U_{10} \\
U_{20} \\
U_{30}
\end{pmatrix}
\]

\[
\begin{pmatrix}
U_{00} \\
U_{10} \\
U_{20} \\
U_{30}
\end{pmatrix} = \begin{pmatrix}
\Pi_{01} \\
& \Pi_{11}
\end{pmatrix} \begin{pmatrix}
L_{01} & L_{11}
\end{pmatrix} \begin{pmatrix}
U_{01} \\
U_{11}
\end{pmatrix} \begin{pmatrix}
\Pi_{02} L_{02} U_{02}
\end{pmatrix}
Obvious generalization of TSQR to LU

- **Block parallel pivoting:**
  - uses a binary tree and is optimal in the parallel case
  
  $$W = \begin{bmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \end{bmatrix} \rightarrow U_{00} \rightarrow U_{01} \rightarrow U_{02}$$

- **Block pairwise pivoting:**
  - uses a flat tree and is optimal in the sequential case
  - introduced by Barron and Swinnerton-Dyer, 1960: block LU factorization used to solve a system with 100 equations on EDSAC 2 computer using an auxiliary magnetic-tape
  - used in PLASMA for multicore architectures and FLAME for out-of-core algorithms and for multicore architectures
  
  $$W = \begin{bmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \end{bmatrix} \rightarrow U_{00} \rightarrow U_{01} \rightarrow U_{02} \rightarrow U_{03}$$
Stability of the LU factorization

- The backward stability of the LU factorization of a matrix $A$ of size $n$-by-$n$
  \[
  \|\hat{L} \cdot \hat{U}\|_\infty \leq (1 + 2(n^2 - n) g_w) \|A\|_\infty
  \]
  depends on the growth factor

  \[
  g_w = \frac{\max_{i,j,k} |a_{ij}^k|}{\max_{i,j} |a_{ij}|}
  \]
  where $a_{ij}^k$ are the values at the $k$-th step.

- $g_w \leq 2^{n-1}$, attained for Wilkinson matrix
  but in practice it is on the order of $n^{2/3} - n^{1/2}$

- Two reasons considered to be important for the average case stability [Trefethen and Schreiber, 90] :
  - the multipliers in $L$ are small,
  - the correction introduced at each elimination step is of rank 1.
Block parallel pivoting

average growth factor (partial pivoting; b=1,2,4,8,16,32)

- Unstable for large number of processors P
- When P=number rows, it corresponds to parallel pivoting, known to be unstable (Trefethen and Schreiber, 90)
Block pairwise pivoting

- Results shown for random matrices
- Will become unstable for large matrices

\[ W = \begin{bmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \end{bmatrix} \rightarrow U_{00} \rightarrow U_{01} \rightarrow U_{02} \rightarrow U_{03} \]
Tournament pivoting - the overall idea

• At each iteration of a block algorithm

\[ A = \begin{pmatrix} \tilde{A}_{11} & \tilde{A}_{21} \\ A_{21} & A_{22} \end{pmatrix} \] \quad b \quad n - b

where

\[ W = \begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix} \]

• Preprocess W to find at low communication cost good pivots for the LU factorization of W, return a permutation matrix P.
• Permute the pivots to top, ie compute PA.
• Compute LU with no pivoting of W, update trailing matrix.

\[ PA = \begin{pmatrix} L_{11} & \quad U_{11} & U_{12} \\ L_{21} & I_{n-b} & A_{22} - L_{21}U_{12} \end{pmatrix} \]
Tournament pivoting for a tall skinny matrix

1) Compute GEPP factorization of each $W_i$, find permutation $\Pi_0$

$$W = \begin{pmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \end{pmatrix} = \begin{pmatrix} \Pi_{00}L_{00}U_{00} \\ \Pi_{10}L_{10}U_{10} \\ \Pi_{20}L_{20}U_{20} \\ \Pi_{30}L_{30}U_{30} \end{pmatrix}$$

Pick b pivot rows, form $A_{00}$
Same for $A_{10}$
Same for $A_{20}$
Same for $A_{30}$

2) Perform $\log_2(P)$ times GEPP factorizations of 2b-by-b rows, find permutations $\Pi_1, \Pi_2$

$$\begin{pmatrix} A_{00} \\ A_{10} \\ A_{20} \\ A_{30} \end{pmatrix} = \begin{pmatrix} \Pi_{01}L_{01}U_{01} \\ \Pi_{11}L_{11}U_{11} \end{pmatrix}$$

Pick b pivot rows, form $A_{01}$
Same for $A_{11}$

$$\begin{pmatrix} A_{01} \\ A_{11} \end{pmatrix} = \frac{\Pi_{02}L_{02}U_{02}}{\Pi_2}$$

3) Compute LU factorization with no pivoting of the permuted matrix:

$$\Pi_2^T \Pi_1^T \Pi_0^T W = LU$$
Tournament pivoting

\[
\begin{align*}
\begin{bmatrix}
W_0 \\
2 & 4 \\
0 & 1 \\
2 & 0 \\
1 & 2 \\
\end{bmatrix} &= \Pi_0 L_0 U_0
\end{align*}
\]

Good pivots for factorizing \( W \)

\[
\begin{align*}
\begin{bmatrix}
P_0 \\
W_0 \\
2 & 4 \\
0 & 1 \\
2 & 0 \\
1 & 2 \\
\end{bmatrix}
&= \begin{bmatrix}
\Pi_0^T W_0 \\
2 & 4 \\
2 & 0 \\
4 & 1 \\
2 & 0 \\
\end{bmatrix} = \Pi_0 L_0 U_0
\end{align*}
\]

\[
\begin{align*}
\begin{bmatrix}
P_1 \\
W_1 \\
2 & 0 \\
0 & 0 \\
4 & 1 \\
1 & 0 \\
\end{bmatrix}
&= \Pi_1 L_1 U_1
\end{align*}
\]

\[
\begin{align*}
\begin{bmatrix}
P_2 \\
W_2 \\
0 & 1 \\
1 & 4 \\
0 & 0 \\
0 & 2 \\
\end{bmatrix}
&= \Pi_2 L_2 U_2
\end{align*}
\]

\[
\begin{align*}
\begin{bmatrix}
P_3 \\
W_3 \\
2 & 1 \\
0 & 2 \\
1 & 0 \\
4 & 2 \\
\end{bmatrix}
&= \Pi_3 L_3 U_3
\end{align*}
\]
• Random matrices from a normal distribution
• Same behaviour for all matrices in our test, and $|L| \leq 4.2$
Stability of CALU (experimental results)

- Results show $\|PA-LU\|/\|A\|$, normwise and componentwise backward errors, for random matrices and special ones
  - See [LG, Demmel, Xiang, SIMAX 2011] for details
  - BCALU denotes binary tree based CALU and FCALU denotes flat tree based CALU
Our “proof of stability” for CALU

• CALU as stable as GEPP in following sense:
  In exact arithmetic, CALU process on a matrix A is equivalent to GEPP process on a larger matrix G whose entries are blocks of A and zeros.

• Example of one step of tournament pivoting:

\[
A = \begin{pmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22} \\
A_{31} & A_{32}
\end{pmatrix}
\]

\[
G = \begin{pmatrix}
\overline{A}_{11} & \overline{A}_{12} \\
A_{21} & A_{21} \\
-A_{31} & A_{32}
\end{pmatrix}
\]

proof possible by using original rows of A during tournament pivoting (not the computed rows of U).
LU factorization and low rank matrices

- For low rank matrices, the factorization of $A_1$ computed as following might not be stable

  Compute $PA = LU$ by using GEPP
  \[ L(k+1:end,k) = A(k+1:end,k)/A(k,k) \]
  Permute the matrix $A_1 = PA$
  Compute $LU$ with no pivoting $A_1 = L_1U_1$
  \[ L(k+1:end,k) = L(k+1:end,k) \times (1/A(k,k)) \]

- Example $A = \text{randn}(6,3)\times \text{randn}(3,5)$, $\max(|\text{abs}(L)|) = 1$, $\max(|\text{abs}(L_1)|) = 10^{15}$

After 4 steps of factorization of $PA$ we obtain:

$PA^4 = \begin{pmatrix}
1.0000 & 0.1729 & 0.6061 & 0.5776 & 0.4789 & -0.3264 \\
0.1729 & 1.0000 & 0.8608 & 0.0543 & -0.2877 & -0.7514 \\
0.6061 & 0.8608 & 1.0000 & 0.3264 & -0.1545 & -0.4597 \\
0.5776 & 0.0543 & 0.3264 & 1.0000 & 2.3333 & 1.7778 \\
0.4789 & -0.2877 & -0.1545 & 2.3333 & 2.3\times10^{-16} & 8.3\times10^{-17} \\
-0.3264 & -0.7514 & -0.4597 & 1.7778 & 2.3\times10^{-16} & 8.3\times10^{-17}
\end{pmatrix}$

\[
\begin{pmatrix}
4.4766 & 3.0163 & -4.7390 & 4.2180 & -0.8164 \\
-1.5439 & -0.4703 & 1.9267 & 1.0925 \\
1.6149 & 2.3623 & 0.3167 \\
\end{pmatrix}
\begin{pmatrix}
9.9\times10^{-16} & 1.6\times10^{-16} & 1
\end{pmatrix}
\]

Schur complement after 4 elimination steps

\[
\begin{pmatrix}
4.4766 & 3.0163 & -4.7390 & 4.2180 & -0.8164 \\
-1.5439 & -0.4703 & 1.9267 & 1.0925 \\
1.6149 & 2.3623 & 0.3167 \\
\end{pmatrix}
\begin{pmatrix}
9.9\times10^{-16} & 1.6\times10^{-16} & 1
\end{pmatrix}
\]

Page 57
LU_PRRP: LU with panel rank revealing pivoting

- Pivots are selected by using strong rank revealing QR on each panel
- The factorization after one panel elimination is written as

\[
PA = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} I_b & 0 \\ A_{21}A_{11}^{-1} & I_{n-b} \end{pmatrix} \begin{pmatrix} A_{11} & A_{12} \\ A_{22} - A_{21}A_{11}^{-1}A_{12} \end{pmatrix}
\]

\(A_{21}A_{11}^{-1}\) is computed through strong rank revealing QR and \(\max(||A_{21}A_{11}^{-1}||)_{ij} \leq f\)

- LU_PRRP and CALU_PRRP stable for pathological cases (Wilkinson matrix) and matrices from two real applications (Volterra integral equation - Foster, a boundary value problem - Wright) on which GEPP fails.
Growth factor in exact arithmetic

- Matrix of size \( m \)-by-\( n \), reduction tree of height \( H = \log(P) \).
- \((CA)LU\text{-}PRRP\) select pivots using strong rank revealing QR (A. Khabou, J. Demmel, LG, M. Gu, SIMAX 2013)
- “In practice” means observed/expected/conjectured values.

<table>
<thead>
<tr>
<th></th>
<th>CALU</th>
<th>GEPP</th>
<th>CALU\text{-}PRRP</th>
<th>LU\text{-}PRRP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upper bound</td>
<td>(2^{n(\log(P) + 1)} - 1)</td>
<td>(2^{n-1})</td>
<td>((1+2b)^{(n/b)\log(P)})</td>
<td>((1+2b)^{(n/b)})</td>
</tr>
<tr>
<td>In practice</td>
<td>(n^{2/3} -- n^{1/2})</td>
<td>(n^{2/3} -- n^{1/2})</td>
<td>((n/b)^{2/3} -- (n/b)^{1/2})</td>
<td>((n/b)^{2/3} -- (n/b)^{1/2})</td>
</tr>
</tbody>
</table>

Better bounds

- For a matrix of size \( 10^7 \)-by-\( 10^7 \) (using petabytes of memory)
  \[ n^{1/2} = 10^{3.5} \]
- When will Linpack have to use the QR factorization for solving linear systems?
CALU – a communication avoiding LU factorization

- Consider a 2D grid of \( P \) processors \( P_r \)-by-\( P_c \), using a 2D block cyclic layout with square blocks of size \( b \).

For \( ib = 1 \) to \( n-1 \) step \( b \)

\[ A^{(ib)} = A(ib:n, ib:n) \]

1. Find permutation for current panel using TSLU \( O(n/b \log_2 P_r) \)
2. Apply all row permutations \( \text{pdlaswp} \) \( O(n/b(\log_2 P_c + \log_2 P_r)) \)
   - broadcast pivot information along the rows of the grid
3. Compute panel factorization \( \text{dtrsm} \)
   \[ O(n/b \log_2 P_c) \]
4. Compute block row of \( U \) \( \text{pdtrsm} \)
   - broadcast right diagonal part of \( L \) of current panel
5. Update trailing matrix \( \text{pdgemm} \)
   - broadcast right block column of \( L \)
   - broadcast down block row of \( U \)
LU for General Matrices

• Cost of CALU vs ScaLAPACK’s PDGETRF
  • $n \times n$ matrix on $P^{1/2} \times P^{1/2}$ processor grid, block size $b$
  • Flops: $(2/3)n^3/P + (3/2)n^2b / P^{1/2}$ vs $(2/3)n^3/P + n^2b/P^{1/2}$
  • Bandwidth: $n^2 \log P/P^{1/2}$ vs same
  • Latency: $3n \log P / b$ vs $1.5n \log P + 3.5n \log P / b$

• Close to optimal (modulo log $P$ factors)
  • Assume: $O(n^2/P)$ memory/processor, $O(n^3)$ algorithm,
  • Choose $b$ near $n / P^{1/2}$ (its upper bound)
  • Bandwidth lower bound:
    $\Omega(n^2 /P^{1/2})$ – just log($P$) smaller
  • Latency lower bound:
    $\Omega(P^{1/2})$ – just polylog($P$) smaller
Performance vs ScaLAPACK

• Parallel TSLU (LU on tall-skinny matrix)
  • IBM Power 5
    • Up to 4.37x faster (16 procs, 1M x 150)
  • Cray XT4
    • Up to 5.52x faster (8 procs, 1M x 150)

• Parallel CALU (LU on general matrices)
  • Intel Xeon (two socket, quad core)
    • Up to 2.3x faster (8 cores, 10^6 x 500)
  • IBM Power 5
    • Up to 2.29x faster (64 procs, 1000 x 1000)
  • Cray XT4
    • Up to 1.81x faster (64 procs, 1000 x 1000)

• Details in SC08 (LG, Demmel, Xiang), IPDPS’10 (S. Donfack, LG).
CALU and its task dependency graph

- The matrix is partitioned into blocks of size $T \times b$.
- The computation of each block is associated with a task.
Scheduling CALU’s Task Dependency Graph

- **Static scheduling**
  - Good locality of data
  - Ignores noise

- **Dynamic scheduling**
  - Keeps cores busy
  - Poor usage of data locality
  - Can have large dequeue overhead
Lightweight scheduling

- Emerging complexities of multi- and mani-core processors suggest a need for self-adaptive strategies
  - One example is work stealing
- Goal:
  - Design a tunable strategy that is able to provide a good trade-off between load balance, data locality, and dequeue overhead.
  - Provide performance consistency
- Approach: combine static and dynamic scheduling
  - Shown to be efficient for regular mesh computation [B. Gropp and V. Kale]

<table>
<thead>
<tr>
<th>Data layout/scheduling</th>
<th>Static</th>
<th>Dynamic</th>
<th>Static/(%dynamic)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column Major Layout (CM)</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Block Cyclic Layout (BCL)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>2-level Block Layout (2l-BL)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

S. Donfack, LG, B. Gropp, V. Kale, IPDPS 2012
Lightweight scheduling

• A self-adaptive strategy to provide
  • A good trade-off between load balance, data locality, and dequeue overhead.
  • Performance consistency
  • Shown to be efficient for regular mesh computation [B. Gropp and V. Kale]

Combined static/dynamic scheduling:
• A thread executes in priority its statically assigned tasks
• When no task ready, it picks a ready task from the dynamic part
• The size of the dynamic part is guided by a performance model
Data layout and other optimizations

• Three data distributions investigated
  • CM: Column major order for the entire matrix
  • BCL: Each thread stores contiguously (CM) the data on which it operates
  • 2l-BL: Each thread stores in blocks the data on which it operates

• And other optimizations
  • Updates (dgemm) performed on several blocks of columns (for BCL and CM layouts)
Impact of data layout

Eight socket, six core machine based on AMD Opteron processor (U. of Tennessee).

- **BCL**: Each thread stores contiguously (CM) its data
- **2I-BL**: Each thread stores in blocks its data
Best performance of CALU on multicore architectures

- Reported performance for PLASMA uses LU with block pairwise pivoting.
- GPU data courtesy of S. Donfack
Parallel write avoiding algorithms

Need to avoid writing suggested by emerging memory technologies, as NVMs:
• Writes more expensive (in time and energy) than reads
• Writes are less reliable than reads

Some examples:
• Phase Change Memory: Reads 25 us latency
  Writes: 15x slower than reads (latency and bandwidth)
  consume 10x more energy
• Conductive Bridging RAM - CBRAM
  Writes: use more energy (1pJ) than reads (50 fJ)
• Gap improving by new technologies such as XPoint and other FLASH alternatives, but not eliminated
Parallel write-avoiding algorithms

- Matrix A does not fit in DRAM (of size M), need to use NVM (of size $n^2 / P$)

- Two lower bounds on volume of communication
  - Interprocessor communication: $\Omega \left( \frac{n^2}{P^{1/2}} \right)$
  - Writes to NVM: $\frac{n^2}{P}$

- Result: any three-nested loop algorithm (matrix multiplication, LU,..), must asymptotically exceed at least one of these lower bounds
  - If $\Omega \left( \frac{n^2}{P^{1/2}} \right)$ words are transferred over the network, then $\Omega \left( \frac{n^2}{P^{2/3}} \right)$ words must be written to NVM!

- Parallel LU: choice of best algorithm depends on hardware parameters

<table>
<thead>
<tr>
<th></th>
<th>#words interprocessor comm.</th>
<th>#writes NVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left-looking</td>
<td>$O\left(\frac{n^3 \log^2 P}{P M^{1/2}}\right)$</td>
<td>$O\left(\frac{n^2}{P}\right)$</td>
</tr>
<tr>
<td>Right-looking</td>
<td>$O\left(\frac{n^2 \log P}{P^{1/2}}\right)$</td>
<td>$O\left(\frac{n^2 \log^2 P}{P^{1/2}}\right)$</td>
</tr>
</tbody>
</table>