Mapping Communication-Avoiding QR Decomposition to Various Architectures

Grey Ballard

UC Berkeley

July 16, 2010

Research supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227)
Collaborators

- Michael Anderson, UC Berkeley EECS
- Jim Demmel, UC Berkeley EECS
- Laura Grigori, INRIA
- Mark Hoemmen, Sandia NL
- Julien Langou, U Colorado Denver
- Bebop group, UC Berkeley
- PLASMA group, UT Knoxville
Communication is expensive and should be avoided

“Communication-Avoiding QR” (CAQR) is a new algorithm which moves asymptotically less data than existing QR decomposition algorithms and achieves theoretical lower bounds

When tuned to specific architectures, CAQR achieves performance speedups in practice, especially for tall and skinny matrices
1 Memory and Communication Cost Models

2 Communication-Avoiding QR

3 Architectures and Implementations
   - Out-of-core
   - Grid
   - Multicore
   - GPU
By *communication* we mean

- moving data within memory hierarchy on a sequential computer
- moving data between processors on a parallel computer
Measure communication in terms of *messages* and *words*

- Flop cost: $\gamma$
- Cost of message of size $w$ words: $\alpha + \beta w$
- Total running time of an algorithm (ignoring overlap):
  \[ \alpha \cdot (# \text{ messages}) + \beta \cdot (# \text{ words}) + \gamma \cdot (# \text{ flops}) \]
  
- think of $\alpha$ as latency+overhead cost, $\beta$ as inverse bandwidth

As flop rates continue to improve more quickly than data transfer rates, the relative cost of communication (the first two terms) grows larger
Given an architecture with limited fast/local memory, there exist asymptotic lower bounds for the number of words and messages that must be communicated

- proven for matrix multiplication in [HK81] and [ITT04]
- proven for QR and nearly all direct linear algebra in [BDHS10]

1. CAQR achieves the lower bounds for both sequential and parallel computers

2. Standard algorithm implemented in (Sca)LAPACK [ABB+92, BJCD+97] moves asymptotically more data
## Communication Costs - Sequential

<table>
<thead>
<tr>
<th></th>
<th>LAPACK</th>
<th>CAQR</th>
<th>Lower Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td># words</td>
<td>$O(n^4/M)$</td>
<td>$O(n^3/\sqrt{M})$</td>
<td>$\Omega(n^3/\sqrt{M})$</td>
</tr>
<tr>
<td># messages</td>
<td>$O(n^3/M)$</td>
<td>$O(n^3/M^{3/2})$</td>
<td>$\Omega(n^3/M^{3/2})$</td>
</tr>
</tbody>
</table>

- QR decomposition of square $n \times n$ matrix
- $M$ is the size of the fast memory
## Communication Costs - Parallel

<table>
<thead>
<tr>
<th></th>
<th>ScaLAPACK</th>
<th>CAQR</th>
<th>Lower Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td># words</td>
<td>$O\left(\frac{n^2}{\sqrt{P}} \log P\right)$</td>
<td>$O\left(\frac{n^2}{\sqrt{P}} \log P\right)$</td>
<td>$\Omega\left(\frac{n^2}{\sqrt{P}}\right)$</td>
</tr>
<tr>
<td># messages</td>
<td>$O\left(n \log^2 P\right)$</td>
<td>$O\left(\sqrt{P} \log^3 P\right)$</td>
<td>$\Omega\left(\sqrt{P}\right)$</td>
</tr>
</tbody>
</table>

- QR decomposition of square $n \times n$ matrix
- $P$ is the number of processors
- Local memory size is $O\left(\frac{n^2}{P}\right)$
Memory and Communication Cost Models

Communication-Avoiding QR

Architectures and Implementations
- Out-of-core
- Grid
- Multicore
- GPU
Standard ("right-looking") algorithms work by repeatedly factoring a panel and then updating the trailing matrix:

While the update of the trailing matrix can be done efficiently, the communication bottleneck for Householder QR occurs in the panel factorization – QR decomposition of a tall skinny matrix.
Communication-Avoiding TSQR

New approach ("TSQR" algorithm [DGHL08]) breaks the panel into row blocks and uses one reduction-like operation

The whole factorization looks like

\[
A = \begin{pmatrix}
A_0 \\
A_1 \\
A_2 \\
A_3
\end{pmatrix} = \begin{pmatrix}
Q_{00} & Q_{10} & Q_{20} & Q_{30}
\end{pmatrix} \cdot \begin{pmatrix}
Q_{01} \\
Q_{11}
\end{pmatrix} \cdot Q_{02} \cdot R_{02}
\]

Note: \( Q \) is stored implicitly in different format than Householder QR
Optimality

Using a binary tree minimizes communication in the parallel model

![Binary tree diagram]

Using a flat tree minimizes communication in the sequential model

![Flat tree diagram]

- Any tree is possible—the shape can be mapped to the architecture
- Different shapes yield different implicit storage schemes for $Q$
Memory and Communication Cost Models

Communication-Avoiding QR

Architectures and Implementations
- Out-of-core
- Grid
- Multicore
- GPU
Consider the case of a sequential computer and a matrix which is too big to fit in DRAM (need “out-of-core” algorithm)

- performance bottleneck is data movement between disk and DRAM
Consider the case of a sequential computer and a matrix which is too big to fit in DRAM (need “out-of-core” algorithm)

- performance bottleneck is data movement between disk and DRAM
Using the flat tree, achieved “infinite” speedup over LAPACK’s Householder QR using virtual memory [DGHL08]

- TSQR performance was only 2× slower on a PowerPC laptop than as though DRAM were infinite
- LAPACK took too long to measure for matrices that didn’t fit in DRAM (e.g. 67 million rows and 64 columns)

Note: TSQR with a flat tree is the same algorithm as in [GvdG05]
Consider the case of a matrix distributed across multiple clusters in various cities across a country

- e.g. Grid’5000 in France
- huge differences in communication costs between intra-cluster and inter-cluster data movement
Consider the case of a matrix distributed across multiple clusters in various cities across a country

- e.g. Grid’5000 in France
- huge differences in communication costs between intra-cluster and inter-cluster data movement
Using a hierarchy of binary trees, TSQR consistently outperformed ScaLAPACK’s Householder QR [ACD+10]

- TSQR was faster than ScaLAPACK on one site and achieved linear speedup over four sites (more than $4 \times$ speedup on matrix of size $10,000,000 \times 64$)
- ScaLAPACK did not scale over multiple sites, as it cannot avoid inter-cluster communication
Consider the case of a shared memory computer (multi- or many-core) where the matrix lives in DRAM

- sequential model fits each core independently
- parallel model fits inter-core communication
Consider the case of a shared memory computer (multi- or many-core) where the matrix lives in DRAM

- sequential model fits each core independently
- parallel model fits inter-core communication
Multicore

Consider the case of a shared memory computer (multi- or many-core) where the matrix lives in DRAM

- sequential model fits each core independently
- parallel model fits inter-core communication
Using a flat tree for each core and a binary reduction among cores, TSQR outperformed LAPACK’s QR using multithreaded BLAS [MHDY09]

- TSQR with 8 threads achieved over $4 \times$ speedup over LAPACK for matrix of size $1,000,000 \times 10$
- LAPACK did not scale over multiple cores, best performance was single-threaded
The “Parallel Linear Algebra for Scalable Multicore Architectures” project aims to provide LAPACK functionality with parallel performance using “tiled” algorithms and DAG scheduling.

To improve performance on tall-skinny QR, PLASMA has adopted the hybrid tree shape for panel factorizations:

- This is inspired by the communication-avoiding approach, but in fact locality within a flat tree is not enforced.
- Performance gains for PLASMA are due to greater parallelism.

PLASMA attains over $8 \times$ speedup over multithreaded MKL on a matrix of size $51200 \times 200$ [HLAD09].
Consider the case of a GPU where the matrix lives in global memory:
- limited on-chip “shared” memory
- many data-parallel threads available to work on local problem
Consider the case of a GPU where the matrix lives in global memory

- limited on-chip “shared” memory
- many data-parallel threads available to work on local problem
Ongoing work by Michael Anderson uses a binary tree
- achieves up to $11 \times$ speedup over GPU-only implementation of Householder QR on a matrix of size $8192 \times 32$
- achieves $2 \times$ speedup for video processing (robust PCA) application matrix of size $64384 \times 100$

Best speedup over multithreaded MKL on eight cores is $5 \times$
Conclusions and Future Work

- Communication is an important metric for performance

- CAQR provides flexibility in reduction tree for tuning to different architectures
  - practical speedups in many different situations
  - largest improvements for very skinny matrices

- Other architectures?
  - preliminary work on the cloud

- Other algorithms?
  - TSQR used as a kernel in many scenarios (GMRES, randomized eigensolver, two-sided factorizations)
Thank you!
References I

*LAPACK Users’ Guide.* 
Also available from http://www.netlib.org/lapack/.

E. Agullo, C. Coti, J. Dongarra, T. Herault, and J. Langou.  
QR factorization of tall and skinny matrices in a grid computing environment.  

E. Agullo, J. Dongarra, Bilel Hadri, Jakub Kurzak, Julie Langou, Julien Langou, Hatem Ltaief, Piotr Luszczek, and Asim YarKhan.  

Minimizing communication in linear algebra.  

*
ScaLAPACK Users’ Guide.*  
Also available from http://www.netlib.org/scalapack/.

J. Demmel, L. Grigori, M. Hoemmen, and J. Langou.  
Implementing communication-optimal parallel and sequential QR factorizations.  
Brian C. Gunter and Robert A. van de Geijn.
Parallel out-of-core computation and updating of the QR factorization.

J. W. Hong and H. T. Kung.
I/O complexity: The red-blue pebble game.

B. Hadri, H. Ltaief, E. Agullo, and J. Dongarra.
Enhancing parallelism of tile qr factorization for multicore architectures.
Submitted to *Transactions on Parallel and Distributed Systems*, 2009.
Available at http://www.netlib.org/lapack/lawnspdf/lawn222.pdf.

D. Irony, S. Toledo, and A. Tiskin.
Communication lower bounds for distributed-memory matrix multiplication.

M. Mohiyuddin, M. Hoemmen, J. Demmel, and K. Yelick.
Minimizing communication in sparse matrix solvers.
Householder QR works column-by-column, applying orthogonal transformations of the form

$$(I - \tau uu^T)A = A - \tau u(u^T A)$$

For each column,

1. compute Householder vector (requires computing norm of the column)
2. compute matrix-vector product
3. perform rank-one update of trailing matrix

(Q is stored implicitly as a set of Householder vectors)
## Grid - latencies

<table>
<thead>
<tr>
<th>Latency (ms)</th>
<th>Orsay</th>
<th>Toulouse</th>
<th>Bordeaux</th>
<th>Sophia</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orsay</td>
<td>0.07</td>
<td>7.97</td>
<td>6.98</td>
<td>6.12</td>
</tr>
<tr>
<td>Toulouse</td>
<td>0.03</td>
<td>9.03</td>
<td>8.18</td>
<td></td>
</tr>
<tr>
<td>Bordeaux</td>
<td></td>
<td>0.05</td>
<td>7.18</td>
<td></td>
</tr>
<tr>
<td>Sophia</td>
<td></td>
<td></td>
<td></td>
<td>0.06</td>
</tr>
</tbody>
</table>
Source: Jack Dongarra
Source: Jack Dongarra
Source: Jack Dongarra
Source: Jack Dongarra
Source: Jack Dongarra
Source: Jack Dongarra
Source: Jack Dongarra
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>_</th>
<th></th>
<th></th>
<th>_</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Source: Jack Dongarra
Source: Jack Dongarra
Source: Jack Dongarra
Source: Jack Dongarra
Source: Jack Dongarra