Scheduling of QR Factorization Algorithms on SMP and Multi-core Architectures

Gregorio Quintana-Ortí  Enrique S. Quintana-Ortí
Ernie Chan Robert A. van de Geijn Field G. Van Zee
quintana@icc.uji.es

Universidad Jaime I de Castellón (Spain)
The University of Texas at Austin

16th EuroMicro PDP, 2008
New dense linear algebra libraries for multicore processors

- Scalability for manycore
- Data locality
- Heterogeneity?
Motivation

LAPACK (*Linear Algebra Package*)

- Fortran-77 codes
- One routine (algorithm) per operation in the library
- Storage in column major order

- Parallelism extracted from calls to multithreaded BLAS

- Extracting parallelism only from BLAS limits the scalability of the solution!
- Column major order does hurt data locality
Motivation

**FLAME (Formal Linear Algebra Methods Environment)**

- Libraries of algorithms, not codes
- Notation reflects the algorithm
- APIs to transform algorithms into codes
- Systematic derivation procedure (automated using Mathematica)
- Storage and algorithm are independent

- Parallelism dictated by data dependencies, extracted at execution time
- Storage-by-blocks
Outline

1 Motivation
2 Basic QR
3 Practical QR
4 Parallelization
5 New algorithm-by-blocks
6 Experimental results
7 Concluding remarks
Outline

1. Motivation
2. Basic QR
   Overview of FLAME
3. Practical QR
4. Parallelization
5. New algorithm-by-blocks
6. Experimental results
7. Concluding remarks
The QR Factorization

Definition

Given $A \rightarrow m \times n$, $m \geq n$,

$$A = QR$$

with $Q \rightarrow m \times m$ orthogonal, $R \rightarrow m \times n$ upper triangular.

Interest

- Solution of linear systems $Ax = b$
- Solution of linear-least squares problems $\min \|Ax - b\|$

Computation via Householder reflectors

Given $x \not\equiv 0$, there exist a Householder reflector, defined by $[u, \eta, \beta] := h(x)$, such that all entries of $h(x)x$ except the first one equal zero.
The QR Factorization: Whiteboard Presentation

\[ A \rightarrow \text{done} \]

\[ \alpha_{11} : = \eta \]
\[ a_{12}^T : = \eta \]
\[ A_{22} : = \beta_1 w^T \]
\[ a_{21} : = u_2 \]
\[ A_{22} : = \beta_1 u_2 w^T \]

\[ A \rightarrow \text{done} \]

\[ \alpha_{11} \]
\[ a_{12}^T \]
\[ a_{21} \]
\[ A_{22} \]

\[ A \rightarrow \text{done} \]

\[ A \rightarrow \text{done} \]

\[ A \rightarrow \text{done} \]

\[ A \rightarrow \text{done} \]
FLAME Notation

\[
\begin{array}{cc}
\text{done} & \text{done} \\
\text{done} & \text{A (partially updated)} \\
\alpha_{11} & a_{12}^T \\
a_{21} & A_{22}
\end{array}
\]

Repartition

\[
\begin{pmatrix}
A_{TL} & A_{TR} \\
A_{BL} & A_{BR}
\end{pmatrix}
\rightarrow
\begin{pmatrix}
A_{00} & a_{01} & A_{02} \\
a_{10}^T & \alpha_{11} & a_{12}^T \\
A_{20} & a_{21} & A_{22}
\end{pmatrix}
\]

where \( \alpha_{11} \) is a scalar.
Algorithm: $[A, b] := \text{QR}_\text{UNB}(A)$

Partition $A \rightarrow \begin{pmatrix} A_{TL} & A_{TR} \\ A_{BL} & A_{BR} \end{pmatrix}$, $b \rightarrow \begin{pmatrix} b_T \\ b_B \end{pmatrix}$

where $A_{TL}$ is $0 \times 0$, $b_T$ has 0 elements

while $n(A_{BR}) \neq 0$ do

Repartition

$\begin{pmatrix} A_{TL} & A_{TR} \\ A_{BL} & A_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} A_{00} & a_{01} & A_{02} \\ a_{10} & a_{12} & \alpha_{11} \\ A_{20} & a_{21} & A_{22} \end{pmatrix}$, $b \rightarrow \begin{pmatrix} b_0 \\ \beta_1 \\ b_2 \end{pmatrix}$

where $\alpha_{11}$ and $\beta_1$ are scalars

$[u_2, \eta, \beta_1] := h(\alpha_{11}, a_{21})$

$w^T := a_{12}^T + u_2^T A_{22}$

$\begin{pmatrix} \alpha_{11} & a_{12}^T \\ a_{21} & A_{22} \end{pmatrix} := \begin{pmatrix} \eta & a_{12}^T - \beta_1 w^T \\ u_2 & A_{22} - \beta_1 u_2 w^T \end{pmatrix}$

Continue with

$\begin{pmatrix} A_{TL} & A_{TR} \\ A_{BL} & A_{BR} \end{pmatrix} \leftarrow \begin{pmatrix} A_{00} & a_{01} & A_{02} \\ a_{10} & a_{12} & \alpha_{11} \\ A_{20} & a_{21} & A_{22} \end{pmatrix}$, $b \leftarrow \begin{pmatrix} b_0 \\ \beta_1 \\ b_2 \end{pmatrix}$

endwhile
FLAME Code

From algorithm to code...

**FLAME notation**

Repartition

\[
\begin{pmatrix}
A_{TL} & A_{TR} \\
A_{BL} & A_{BR}
\end{pmatrix}
\rightarrow
\begin{pmatrix}
A_{00} & a_{01} & A_{02} \\
\alpha_{10} & \alpha_{11} & \alpha_{12} \\
A_{20} & a_{21} & A_{22}
\end{pmatrix}
\]

where $\alpha_{11}$ is a scalar

**FLAME/C code**

```
FLA_Repart_2x2_to_3x3(
    ATL, /**/ ATR,       &A00,  /**/ &a01,  &A02,
    /* *************** */ /* *********************** */
    &a10t,  /**/ &alpha11, &a12t,
    ABL, /**/ ABR,       &A20,  /**/ &a21,  &A22,
    1, 1, FLA_BR );
```
int FLA_QR_unb( FLA_Obj A, FLA_Obj b )
{
    /* ... FLA_Part_2x2( ); ... */

    while ( FLA_Obj_width( ABR ) != 0 ){
        FLA_Repart_2x2_to_3x3( ATL, /**/ ATR, &A00, /**/ &a01, &A02, /* ************* */ /* ************************** */ &a10t, /**/ &alpha11, &a12t, ABL, /**/ ABR, &A20, /**/ &a21, &A22, 1, 1, FLA_BR );

        /* ... */
        /*-------------------------------------------------------------------*/
        FLA_House( eta, alpha11, a21 ); /* [ u_2, eta, beta_1 ] := 
            h( alpha_11, a21 ) */
        /* alpha_11 := beta_1 */
        /* a21 := u2 */
        FLA_Copy( a12t, wt ); /* wt := -(a12t 
            +u2^t * A22)*/

        FLA_Gemv( FLA_TRANSPOSE, FLA_ONE, A22, u2, FLA_ONE, wt );
        FLA_Axpy( beta1, wt, a12t ); /* a_12t := 
            a_12t + beta_1 * wt */
        FLA_Ger( beta1, u2, wt, A22 ); /* A22 := 
            A22 + beta_1 * u2 * wt */
        /*-----------------------------------------------*/

        /* FLA_Cont_with_3x3_to_2x2( ); ... */
    }
}

Visit http://www.cs.utexas.edu/users/flame/Spark/...

- M-script code for MATLAB: FLAME@lab
- C code: FLAME/C
- Other APIs:
  - Latex
  - Fortran-77
  - LabView
  - Message-passing parallel: PLAPACK
  - FLAG: GPUs
  - FLAOOC: Out-of-Core
Outline

1. Motivation
2. Basic QR
3. Practical QR
   Blocked algorithm and use of BLAS for high-performance
4. Parallelization
5. New algorithm-by-blocks
6. Experimental results
7. Concluding remarks
Blocked Algorithm for High Performance

<table>
<thead>
<tr>
<th>Algorithm:</th>
<th>([A, S] := \text{QR_BLK}(A))</th>
</tr>
</thead>
</table>
| Partition  | \[
\begin{pmatrix}
A_{TL} & A_{TR} \\
A_{BL} & A_{BR}
\end{pmatrix}
\rightarrow
\begin{pmatrix}
A_{00} & A_{01} & A_{02} \\
A_{10} & A_{11} & A_{12} \\
A_{20} & A_{21} & A_{22}
\end{pmatrix},
\begin{pmatrix}
S_T \\
S_B
\end{pmatrix}
\rightarrow
\begin{pmatrix}
S_0 \\
S_1 \\
S_2
\end{pmatrix}
\] |
| where      | \([A_{11}] \) is \(b \times b\), \(S_1\) has \(b\) rows |
| while      | \(n(A_{BR}) \neq 0\) do |
| Determine  | block size \(b\) |
| Repartition| |
| Computed   | \[
\begin{pmatrix}
\frac{A_{11}}{A_{21}} \\
\frac{A_{21}}{A_{22}}
\end{pmatrix},
\begin{pmatrix}
b_1
\end{pmatrix}
\rightarrow
\begin{pmatrix}
\{U|R\}_{11}
\end{pmatrix},
\begin{pmatrix}
b_1
\end{pmatrix}
\] := \(\text{QR\_UNB} \left( \begin{pmatrix}
\frac{A_{11}}{A_{21}}
\end{pmatrix}\right)\) |
| Compute    | \(S_1\) from \(A_{11}, A_{12}, b_1\) |
| \(W\)      | \(\begin{pmatrix}
U_{11}^T & U_{21}^T
\end{pmatrix}
\begin{pmatrix}
A_{12} \\
A_{22}
\end{pmatrix}
\) |
| \(A_{12} \) | := \(\begin{pmatrix}
A_{12} \\
A_{22}
\end{pmatrix} + \begin{pmatrix}U_{11} \\
U_{21}
\end{pmatrix} S_1 W\) |
| Continue   | with |
| \(\ldots\) | |
| endwhile   | |
Blocked Algorithm for High Performance

**LAPACK implementation: kernels in BLAS**

\[
\begin{pmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{pmatrix}, \quad A_{11} \text{ is } b \times b
\]

1. \( QR\_\text{UNB} \left( \begin{array}{c}
A_{11} \\
A_{21}
\end{array} \right) \) \quad \text{Unblk. QR, } O(nb^2) \text{ flops}

2. \( W := \left( \begin{array}{c|c}
U_{11}^T & U_{21}^T
\end{array} \right) \left( \begin{array}{c}
A_{12} \\
A_{22}
\end{array} \right) \) \quad \text{GEMM, } O(nb^2) \text{ flops}

3. \( \left( \begin{array}{c}
A_{12} \\
A_{22}
\end{array} \right) := \left( \begin{array}{c}
A_{12} \\
A_{22}
\end{array} \right) + \left( \begin{array}{c}
U_{11} \\
U_{21}
\end{array} \right) S_1 W \) \quad \text{GEMM, } O(n^2b) \text{ flops}
Outline

1. Motivation
2. Basic QR
3. Practical QR
4. Parallelization
   - Control-flow vs. data-flow parallelism
   - Storage-by-blocks API
5. New algorithm-by-blocks
6. Experimental results
7. Concluding remarks
Parallelization on Shared-Memory Architectures

LAPACK parallelization: kernels in multithread BLAS

\[
\begin{pmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{pmatrix}, \quad A_{11} \text{ is } b \times b
\]

- **Advantage:** Use legacy code
- **Drawbacks:**
  - Each call to BLAS is a synchronization point for threads
  - As the number of threads increases, serial operations with cost \( O(nb^2) \) are no longer negligible compared with \( O(n^2b) \)
Parallelization on Shared-Memory Architectures

FLAME parallelization: SuperMatrix

- Traditional (and pipelined) parallelizations are limited by the control dependencies dictated by the code
- The parallelism should be limited only by the data dependencies between operations!
- In dense linear algebra, imitate a superscalar processor: dynamic detection of data dependencies
The *FLAME runtime system* “pre-executes” the code:

- Whenever a routine is encountered, a pending task is annotated in a global task queue
FLAME Parallelization: SuperMatrix

\[
\begin{pmatrix}
A_{00} & A_{01} & A_{02} \\
A_{10} & A_{11} & A_{12} \\
A_{20} & A_{21} & A_{22}
\end{pmatrix}
\]

\[\text{Runtime} \rightarrow \]

1. \(QR_{-\text{UNB}}\left( \begin{pmatrix} A_{00} \\ A_{10} \\ A_{20} \end{pmatrix} \right)\)
2. \(\begin{pmatrix} A_{01} \\ A_{11} \\ A_{21} \end{pmatrix} := Q_{11}^T \begin{pmatrix} A_{01} \\ A_{11} \\ A_{21} \end{pmatrix}\)
3. \(\begin{pmatrix} A_{02} \\ A_{12} \\ A_{22} \end{pmatrix} := Q_{11}^T \begin{pmatrix} A_{02} \\ A_{12} \\ A_{22} \end{pmatrix}\)
4. \(\ldots\)

**SuperMatrix**

- Once all tasks are annotated, the real execution begins!
- Tasks with all input operands available are runnable; other tasks must wait in the global queue
- Upon termination of a task, the corresponding thread updates the list of pending tasks
FLAME Storage-by-Blocks: FLASH

- Algorithm and storage are independent
- Matrices stored by blocks are viewed as matrices of matrices
- No significative modification to the FLAME codes
Outline

1. Motivation
2. Basic QR
3. Practical QR
4. Parallelization
5. New algorithm-by-blocks
   Expose more parallelism
6. Experimental results
7. Concluding remarks
Algorithm-by-blocks for the QR factorization

\[
\begin{pmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{pmatrix}
\]

- All operations on \( A_{22} \) must wait till \( \frac{A_{11}}{A_{21}} \) is factorized.
- Algorithms-by-blocks for the Cholesky factorization do not present this problem.
- Is it possible to design an algorithm-by-blocks for the QR factorization?
Algorithm-by-blocks for the QR factorization

\[
\begin{pmatrix}
A_{11} & A_{12} & A_{13} \\
A_{21} & A_{22} & A_{23} \\
A_{31} & A_{32} & A_{33}
\end{pmatrix}, \quad A_{ij} \text{ is } t \times t
\]

1. Factorize \( Q_{11} A_{11} = R_{11} \)
2. Apply factor \( Q_{11} \):
   \[
   Q_{11}^T A_{12} \mid Q_{11}^T A_{13}
   \]
3. Factorize \( Q_{21} \left( \frac{A_{11}}{A_{21}} \right) = R_{21} \)
4. Apply factor \( Q_{21} \):
   \[
   Q_{21}^T \left( \frac{A_{12}}{A_{22}} \right) \mid Q_{21}^T \left( \frac{A_{13}}{A_{23}} \right)
   \]
5. Repeat steps 2–4 with \( A_{31} \)
Algorithm-by-blocks for the QR factorization

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix}, \quad A_{ij} \text{ is } t \times t$$

Different from traditional QR factorization

- To obtain high performance a blocked algorithm with block size $b \ll t$, is used in the factorization and application of factors
- To maintain the computational cost, the upper triangular structure of $A_{11}$ is exploited during the factorization
Outline

1. Motivation
2. Basic QR
3. Practical QR
4. Parallelization
5. New algorithm-by-blocks
6. Experimental results
7. Concluding remarks
Experimental Results

## General

<table>
<thead>
<tr>
<th>Platform</th>
<th>Specs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET</td>
<td>CC-NUMA with 16 Intel Itanium-2 processors</td>
</tr>
<tr>
<td>NEUMANN</td>
<td>SMP with 8 dual-core AMD Opteron processors</td>
</tr>
</tbody>
</table>

## Implementations

- LAPACK: LAPACK 3.0 routine `dgetrf` + multithreaded MKL
- MKL: Multithreaded routine `dgetrf` in MKL
- AB: Algorithm-by-blocks + serial MKL + storage-by-blocks
Experimental Results

QR factorization on 16 Intel Itanium 2@1.5GHz

Matrix size

GFLOPS

AB
MKL
LAPACK

PDP’08
QR Factorization algorithms on SMP & multi-core
Motivation
Basic QR
Parallelization
Algorithm-by-blocks
Results
Remarks
Experimental Results

QR factorization on 8 dual AMD Opteron@2.2GHz

- AB
- MKL
- LAPACK

Matrix size vs. GFLOPS for QR factorization on 8 dual AMD Opteron@2.2GHz.
More parallelism is needed to deal with the large number of cores of future architectures and data locality issues: traditional dense linear algebra libraries will have to be rewritten.

- An algorithm-by-blocks is possible for the QR factorization similar to those of Cholesky and QR factorizations.
- The FLAME infrastructure (FLAME/C API, FLASH, and SuperMatrix) reduces the time to take an algorithm from whiteboard to high-performance parallel implementation.
Thanks for your attention!

For more information...
Visit http://www.cs.utexas.edu/users/flame

Support...
- *National Science Foundation* awards CCF-0702714 and CCF-0540926 (ongoing till 2010).
- Spanish CICYT project TIN2005-09037-C02-02.
Related publications


Related Approaches

Cilk (MIT) and CellSs (Barcelona SuperComputing Center)

- General-purpose parallel programming
  - Cilk → irregular problems
  - CellSs → for the Cell B.E.
- High-level language based on OpenMP-like pragmas + compiler + runtime system
- Moderate results for dense linear algebra

PLASMA (UTK – Jack Dongarra)

- Traditional style of implementing algorithms: Fortran-77
- Complicated coding
- Runtime system + ?