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

Motivation

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

QR Factorization algorithms on SMP & multi-core

PDP'08

Motivation

Basic QR

Parallelization

Algorithm-by-blocks

Results

Remarks

Outline

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.

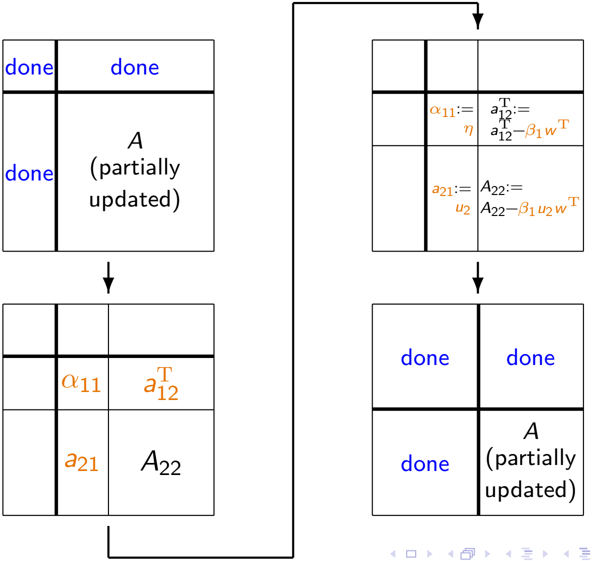
Interest

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

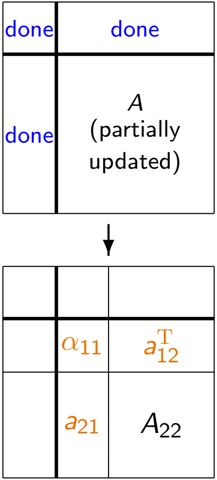
Computation via Householder reflectors

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



FLAME Notation



Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$$

where α_{11} is a scalar

FLAME Notation

Algorithm: $[A,\ b] := \mathrm{QR\_UNB}(A)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$, $b \rightarrow \left(\begin{array}{c} b_T \\ \hline b_B \end{array}\right)$

where A_{TL} is 0×0, b_T has 0 elements

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

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^{\mathrm{T}} & \alpha_{11} & a_{12}^{\mathrm{T}} \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right), \quad b \rightarrow \left(\begin{array}{c} b_0 \\ \hline \beta_1 \\ \hline b_2 \end{array}\right)$$

where α_{11} and β_1 are scalars

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

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

$$\left(\begin{array}{c|c} \alpha_{11} & a_{12}^{\mathrm{T}} \\ \hline a_{21} & A_{22} \end{array}\right) := \left(\begin{array}{c|c} \eta & a_{12}^{\mathrm{T}} - \beta_1 w^{\mathrm{T}} \\ \hline u_2 & A_{22} - \beta_1 u_2 w^{\mathrm{T}} \end{array}\right)$$

Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^{\mathrm{T}} & \alpha_{11} & a_{12}^{\mathrm{T}} \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right), \quad b \leftarrow \left(\begin{array}{c} b_0 \\ \hline \beta_1 \\ \hline b_2 \end{array}\right)$$

endwhile

FLAME Code

From algorithm to code...

FLAME notation

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$$

where α_{11} is a scalar

FLAME/C code

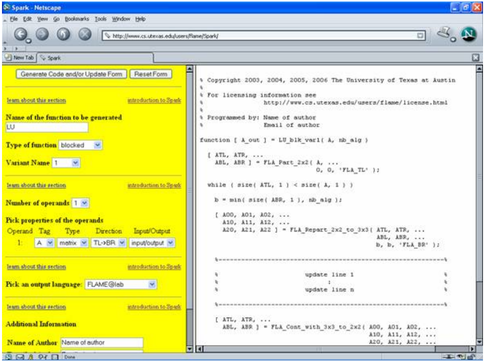
```
FLA_Repart_2x2_to_3x3(
        ATL, /**/ ATR,       &A00,  /**/ &a01,     &A02,
     /* ************* */ /* ************************** */
                             &a10t, /**/ &alpha11, &a12t,
        ABL, /**/ ABR,       &A20,  /**/ &a21,     &A22,
        1, 1, FLA_BR );
```

FLAME Code


```
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( ); ... */
  }
}
```

FLAME Code

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



- M-script code for MATLAB: FLAME@lab
- C code: FLAME/C
- Other APIs:
  - FLaTeX
  - 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

Algorithm: $[A, S] := \mathrm{QR\_BLK}(A)$

Partition ...

where ...

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

Determine block size b

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} S_T \\ \hline S_B \end{array}\right) \rightarrow \left(\begin{array}{c} S_0 \\ \hline S_1 \\ \hline S_2 \end{array}\right)$$

where A_{11} is $b \times b$, S_1 has b rows

$$\left[\left(\frac{A_{11}}{A_{21}}\right), b_1\right] = \left[\left(\frac{\{U \backslash R\}_{11}}{U_{21}}\right), b_1\right] := \mathrm{QR}_{\mathrm{UNB}}\left(\left(\frac{A_{11}}{A_{21}}\right)\right)$$

Compute S_1 from A_{11}, A_{12}, b_1

$$W := \left(U_{11}^{\mathrm{T}} \mid U_{21}^{\mathrm{T}} \right) \left(\frac{A_{12}}{A_{22}}\right)$$

$$\left(\frac{A_{12}}{A_{22}}\right) := \left(\frac{A_{12}}{A_{22}}\right) + \left(\frac{U_{11}}{U_{21}}\right) S_1 W$$

Continue with

...

endwhile

Blocked Algorithm for High Performance

LAPACK implementation: kernels in BLAS

$$\left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right), \quad A_{11} \text{ is } b \times b$$

1. $\mathrm{QR\_UNB}\left(\frac{A_{11}}{A_{21}}\right)$ — Unblk. QR, $O(nb^2)$ flops
2. $W := \left(U_{11}^{\mathrm{T}} \mid U_{21}^{\mathrm{T}} \right) \left(\frac{A_{12}}{A_{22}}\right)$ — GEMM, $O(nb^2)$ flops
3. $\left(\frac{A_{12}}{A_{22}}\right) := \left(\frac{A_{12}}{A_{22}}\right) + \left(\frac{U_{11}}{U_{21}}\right) S_1 W$ — GEMM, $O(n^2 b)$ flops

Outline

Parallelization on Shared-Memory Architectures

LAPACK parallelization: kernels in multithread BLAS

$$\left(\begin{array}{c|c|c} \hline & A_{11} & A_{12} \\ \hline & A_{21} & A_{22} \end{array}\right), \quad A_{11} \text{ is } b \times b$$

- 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 $O(n^2 b)$

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

FLAME Parallelization: SuperMatrix


```
int FLA_QR_blk( FLA_Obj A, FLA_Obj S )
{
  /* ... FLA_Part_2x2( ); ... */

  while ( FLA_Obj_width( ATL ) < FLA_Obj_width( A ) ){

    /* FLA_Repart_2x2_to_3x3( ); ... */

    /*------------------------------------------------------------*/
    FLA_QR_unb( A11,                   /* QR( A11; A21 )          */
                A21, S1 );

    FLA_QR_update_blk( A11, A12,
                       A21, A22, S1 );
            /* W          := (U11^T | U_21^T ) * (A12; A22)
               (A12; A22) := (A12; A22) + (U11; U21) *  S1 * W    */
    /*------------------------------------------------------------*/

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

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

$$\left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right) \xrightarrow{\text{Runtime}}$$

1. $\text{QR\_UNB}\left(\begin{array}{c} A_{00} \\ \hline A_{10} \\ \hline A_{20} \end{array}\right)$
2. $\left(\begin{array}{c} A_{01} \\ \hline A_{11} \\ \hline A_{21} \end{array}\right) := Q_{11}^T \left(\begin{array}{c} A_{01} \\ \hline A_{11} \\ \hline A_{21} \end{array}\right)$
3. $\left(\begin{array}{c} A_{02} \\ \hline A_{12} \\ \hline A_{22} \end{array}\right) := Q_{11}^T \left(\begin{array}{c} A_{02} \\ \hline A_{12} \\ \hline A_{22} \end{array}\right)$
4. ...

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

$$\left(\begin{array}{c|c|c} & & \\ \hline & A_{11} & A_{12} \\ \hline & A_{21} & A_{22} \end{array}\right), \quad A_{11} \text{ is } b \times b$$

- All operations on A_{22} must wait till $\left(\frac{A_{11}}{A_{21}}\right)$ 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

QR Factorization algorithms on SMP & multi-core

PDP'08

$$\left(\begin{array}{c|c|c|c} & & & \\ \hline & A_{11} & A_{12} & A_{13} \\ \hline & A_{21} & A_{22} & A_{23} \\ \hline & A_{31} & A_{32} & A_{33} \end{array}\right), \quad A_{ij} \text{ is } t \times t$$

1. Factorize $Q_{11} A_{11} = R_{11}$
2. Apply factor Q_{11}:
$$Q_{11}^T A_{12} \;\Big|\; Q_{11}^T A_{13}$$
3. Factorize $Q_{21} \left(\frac{A_{11}}{A_{21}}\right) = R_{21}$,
4. Apply factor Q_{21}:
$$Q_{21}^T \left(\frac{A_{12}}{A_{22}}\right) \;\Big|\; Q_{21}^T \left(\frac{A_{13}}{A_{23}}\right)$$
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

Experimental Results

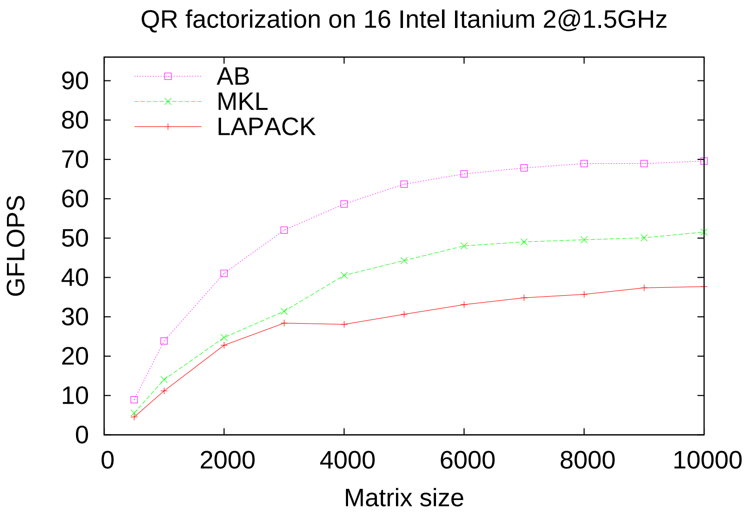
General

Platform	Specs.
SET	CC-NUMA with 16 Intel Itanium-2 processors
NEUMANN	SMP with 8 dual-core AMD Opteron processors

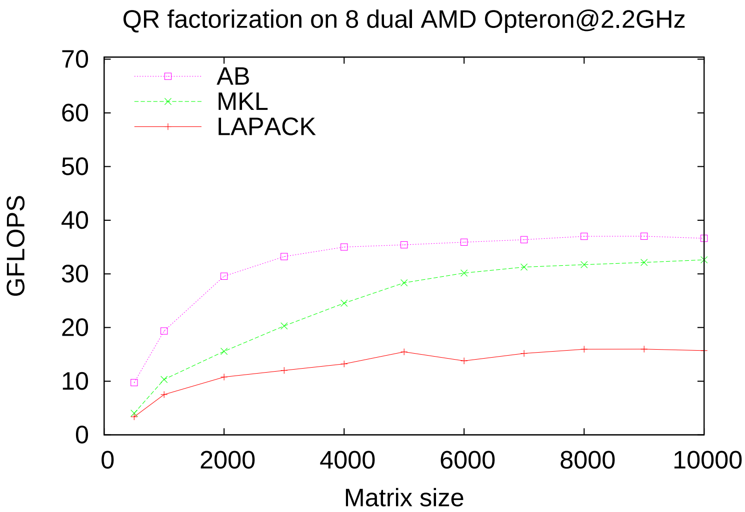
Implementations

- 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



Experimental Results



Concluding Remarks

- More parallelism is needed to deal with the large number of cores of future architectures and data locality issued: 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

- E. Chan, E.S. Quintana-Ortí, G. Quintana-Ortí, R. van de Geijn. SuperMatrix out-of-order scheduling of matrix operations for SMP and multicore architectures. *19th ACM Symp. on Parallelism in Algorithms and Architectures – SPAA'2007.*
- E. Chan, F. Van Zee, R. van de Geijn, E.S. Quintana-Ortí, G. Quintana-Ortí. Satisfying your dependencies with SuperMatrix. *IEEE Cluster 2007*.
- E. Chan, F.G. Van Zee, P. Bientinesi, E.S. Quintana-Ortí, G. Quintana-Ortí, R. van de Geijn. SuperMatrix: A multithreaded runtime scheduling system for algorithms-by-blocks. *Principles and Practices of Parallel Programming – PPoPP'2008.*
- Brian Gunter, R. van de Geijn. Parallel OOC computation and updating of the QR factorization. *ACM Trans. on Mathematical Software*, 31(1):60-78, 2005.

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 pramas + compiler + runtime system
- Moderate results for dense linear algebra

PLASMA (UTK – Jack Dongarra)

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