Design of Scalable Dense Linear Algebra Libraries for Multithreaded Architectures: the LU Factorization

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

Workshop on Programmability Issues for Multi-Core Computers, 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 amount of parallelism and, therefore, the scalability of the solution!
- Column major order does hurt data locality

Motivation

FLAME (*Formal Linear Algebra Methods Environment*)

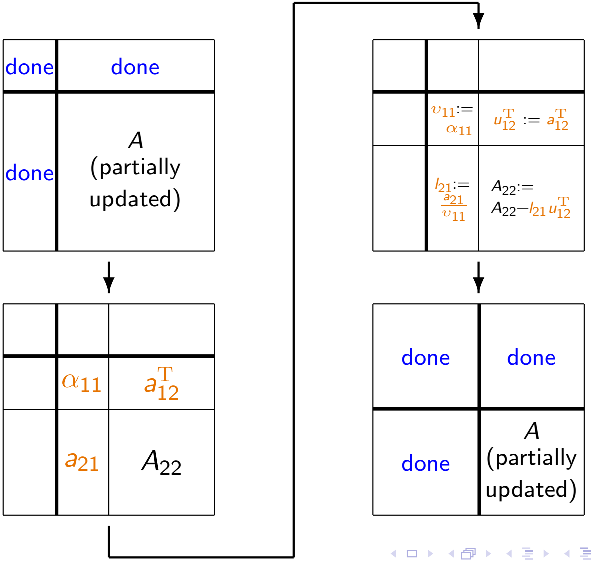
- Libraries of algorithms, not codes
- Notation reflects the algorithm
- APIs to transform algorithms into codes
- Systematic derivation procedure (automated using MATHEMATICA)
- Storage and algorithm are independent

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

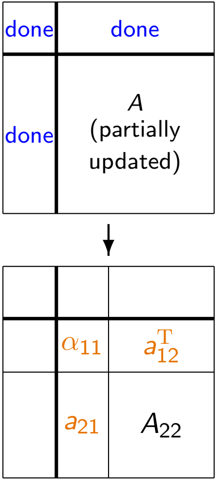
Outline

Outline

The LU 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: $A := \text{LU\_UNB}(A)$

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

where A_{TL} is 0×0

while $n(A_{TL}) < n(A)$ **do**

Repartition

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

where α_{11} is a scalar

$a_{21} := a_{21} / \alpha_{11}$

$A_{22} := A_{22} - a_{21} a_{12}^T$

Continue with

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

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

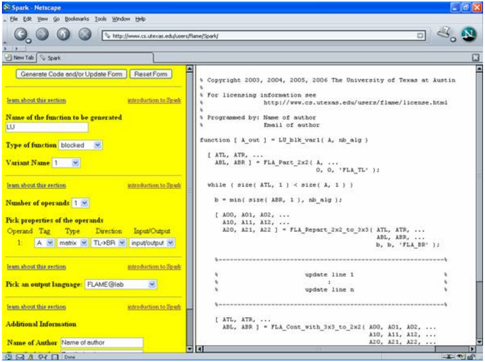
```
FLA_Repart_2x2_to_3x3(
        ATL, /**/ ATR,       &A00,  /**/ &a01,     &A02,
     /* ************* */ /* ************************** */
                             &a10t, /**/ &alpha11, &a12t,
        ABL, /**/ ABR,       &A20,  /**/ &a21,     &A22,
        1, 1, FLA_BR );
```

FLAME Code


```
int FLA_LU_unb( FLA_Obj A )
{
  /* ... FLA_Part_2x2( ); ... */

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

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

    /*------------------------------------------------------------*/
    FLA_Inv_Scal( alpha11, a21 );      /* a21 := a21 / alpha11    */
    FLA_Ger     ( FLA_MINUS_ONE,
                  a21, a12t, A22 );    /* A22 := A22 - a21 * a12t */
    /*------------------------------------------------------------*/

    /* 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
  - GLAME: GPUs
  - FLAOO: Out-of-Core

Outline

Partial Pivoting

Algorithm: $[A, p] := \text{LUP\_UNB}(A)$

Partition ...
where ...
while $n(A_{TL}) < n(A)$ **do**
Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} p_T \\ \hline p_B \end{array}\right) \rightarrow \left(\begin{array}{c} p_0 \\ \hline \pi_1 \\ \hline p_2 \end{array}\right)$$

where α_{11} is 1×1 , π_1 has 1 row

$$\left[\left(\begin{array}{c} \alpha_{11} \\ \hline a_{21} \end{array}\right), \pi_1\right] := \text{Pivot}\left(\begin{array}{c} \alpha_{11} \\ \hline a_{21} \end{array}\right)$$

$$\left(\begin{array}{c|c} a_{10}^T & a_{12}^T \\ \hline A_{20} & A_{22} \end{array}\right) := P(\pi_1)\left(\begin{array}{c|c} a_{10}^T & a_{12}^T \\ \hline A_{20} & A_{22} \end{array}\right)$$

$$a_{21} := a_{21}/\alpha_{11}$$

$$A_{22} := A_{22} - a_{21}a_{12}^T$$

Continue with
...
endwhile

Blocked Algorithm for High Performance

Algorithm: $[A, p] := \text{LUP\_BLK}(A)$

Partition ...
  where ...
while $n(A_{TL}) < n(A)$ **do**
  Determine block size b
  Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right), \left(\begin{array}{c} p_T \\ \hline p_B \end{array}\right) \rightarrow \left(\begin{array}{c} p_0 \\ \hline p_1 \\ \hline p_2 \end{array}\right)$$

  where A_{11} is $b \times b$, p_1 has b rows

$$\left[\left(\begin{array}{c} A_{11} \\ \hline A_{21} \end{array}\right), p_1\right] := \text{LUP\_UNB}\left(\begin{array}{c} A_{11} \\ \hline A_{21} \end{array}\right)$$

$$\left(\begin{array}{c|c} A_{10} & A_{12} \\ \hline A_{20} & A_{22} \end{array}\right) := P(p_1)\left(\begin{array}{c|c} A_{10} & A_{12} \\ \hline A_{20} & A_{22} \end{array}\right)$$

$$A_{12} := \text{TRILU}(A_{11})^{-1} A_{12}$$

$$A_{22} := A_{22} - A_{21} A_{12}$$

  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. $\text{LUP\_UNB}\left(\frac{A_{11}}{A_{21}}\right)$ — Unblocked LU, $O(nb^2)$ flops
2. $A_{12} := \text{TRILU}(A_{11})^{-1} A_{12}$ — TRSM, $O(nb^2)$ flops
3. $A_{22} := A_{22} - A_{21} A_{12}$ — GEMM, $O(n^2 b)$ flops

Outline

1. Motivation
2. Basic LU
3. Practical LU
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

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

- 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 $O(n^2 b)$

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

FLAME Parallelization: SuperMatrix


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

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

    /* FLA_Repart_2x2_to_3x3( ); ... */

    /*------------------------------------------------------------*/
    FLA_LUP_unb( A11, A21, p1 );     /* LU( A11; A21 )            */
    FLA_Apply_pivots( A10, A20,
                      A12, A22, p1 );/* Pivot remaining columns   */
    FLA_Trsm_llnu   ( A11, A12 );    /* A12 := TRILU(A11)^-1 * A12*/
    FLA_Gemm_nn     ( -1, A21, A12,
                       1, A22 );     /* A22 := A22 - A21 * A12    */
    /*------------------------------------------------------------*/

    /* 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

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

1. $\text{LUP\_UNB}\left(\begin{array}{c} A_{00} \\ \hline A_{10} \\ \hline A_{20} \end{array}\right)$
2. $A_{01} := \text{TRILU}(A_{00})^{-1} A_{01}$
3. $A_{02} := \text{TRILU}(A_{00})^{-1} A_{02}$
4. $A_{11} := A_{11} - A_{10} A_{01}$
5. ...

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

Algorithm-by-blocks for the LU factorization

- Pivoting for stability limits the amount of parallelism

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

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 LU factorization while maintaining pivoting?

Algorithm-by-blocks for the LU factorization: LU factorization with incremental pivoting

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

1. Factorize $P_{11} A_{11} = L_{11} U_{11}$
2. Apply permutation P_{11} and factor L_{11}:
$$L_{11}^{-1} P_{11} A_{12} \mid L_{11}^{-1} P_{11} A_{13}$$
3. Factorize $P_{21} \left(\frac{A_{11}}{A_{21}}\right) = L_{21} U_{21}$,
4. Apply permutation P_{21} and factor L_{21}:
$$L_{21}^{-1} P_{21} \left(\frac{A_{12}}{A_{22}}\right) \Bigg| L_{21}^{-1} P_{21} \left(\frac{A_{13}}{A_{23}}\right)$$
5. Repeat steps 2–4 with A_{31}

Algorithm-by-blocks for the LU factorization: LU factorization with incremental pivoting

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

Different from LU factorization with column pivoting

- To preserve structure, permutations only applied to blocks on the right!
- 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 Intel Pentium4 processors

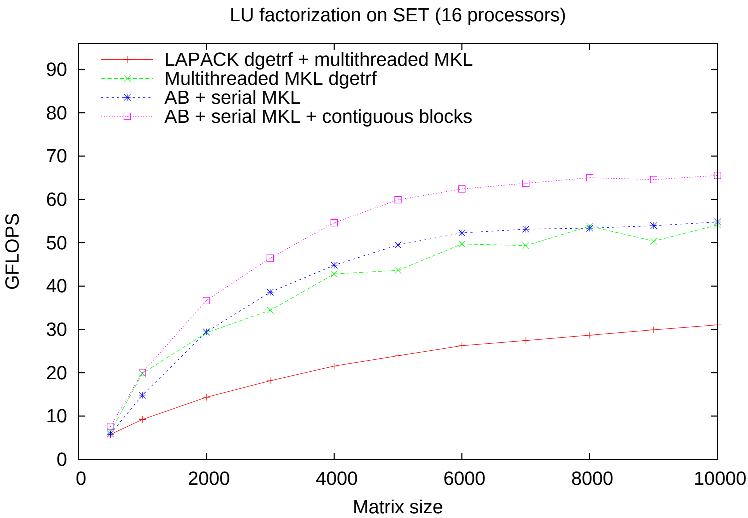
Implementations

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

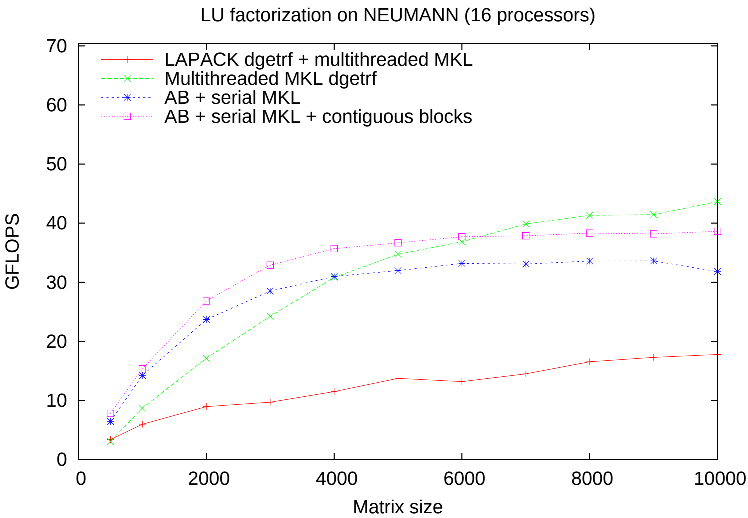
Experimental Results

Scalable Dense Linear Algebra Libraries: LU Factorization

PPAM'07



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 LU 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.*
- E.S. Quintana-Ortí, R. van de Geijn. Updating an LU factorization with pivoting. *ACM Trans. on Mathematical Software, 2008.*

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 + ?