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

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

Workshop on Multi-Threaded Architectures and Applications
Miami, 2008
New dense linear algebra libraries for multicore processors

- Scalability for manycore
- Data locality
- Heterogeneity?
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

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

Overview of FLAME

Practical LU

Parallelization

New algorithm-by-blocks

Experimental results

Concluding remarks
The LU Factorization: Whiteboard Presentation

\[
\begin{align*}
\alpha_{11} &:= u_{11} \\
a_{21} &:= u_{12} \\
A_{22} &:= a_{22} - l_{21} \cdot u_{12}
\end{align*}
\]
FLAME Notation

Motivation
Basic LU
Practical LU
Parallelization
Algorithm-by-blocks
Results
Remarks

Scalable Dense Linear Algebra Libraries: LU Factorization

MTAAP'08

Repartition

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

where \( \alpha_{11} \) is a scalar
Algorithm: $A := LU\_UNB(A)$

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

where $A_{TL}$ is $0 \times 0$

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

Repartition

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

where $\alpha_{11}$ is a scalar

\[
a_{21} := a_{21} / \alpha_{11}
\]

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

Continue with

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

endwhile
FLAME Code

From algorithm to code...

**FLAME notation**

<table>
<thead>
<tr>
<th>Repartition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A_{TL} )</td>
</tr>
<tr>
<td>( A_{BL} )</td>
</tr>
</tbody>
</table>

\[
\begin{pmatrix}
A_{TL} & A_{TR} \\
A_{BL} & A_{BR}
\end{pmatrix} \rightarrow \begin{pmatrix}
A_{00} & a_{01} & A_{02} \\
a_{T0} & \alpha_{11} & a_{T12} \\
A_{20} & a_{21} & A_{22}
\end{pmatrix}
\]

*where* \( \alpha_{11} \) *is a scalar*

**FLAME/C code**

```c
FLA_Repart_2x2_to_3x3(
    ATL, /**</ ATR,
    &A00, /**</ &a01,
    &A02,
    /* *************** */ /* ************ */
    &aT0, /**</ &alpha11, &a12t,
    ABL, /**</ ABR,
    &A20, /**</ &a21, &A22,
    1, 1, FLA_BR);
```
```c
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 */

        /*---------------------------------------------*/

    }
}
```
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
  - FLAOO: Out-of-Core
Outline

1. Motivation
2. Basic LU
3. Practical LU
   - Pivoting for stability
   - Blocked algorithm and use of BLAS for high-performance
4. Parallelization
5. New algorithm-by-blocks
6. Experimental results
7. Concluding remarks
Partial Pivoting

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

<table>
<thead>
<tr>
<th>Partition</th>
<th>(\cdots)</th>
</tr>
</thead>
<tbody>
<tr>
<td>where</td>
<td>(\cdots)</td>
</tr>
</tbody>
</table>
| while \(n(A_{TL}) < n(A)\) do  
| Repartition |  
| \[
\begin{pmatrix}
A_{TL} & A_{TR} \\
A_{BL} & A_{BR}
\end{pmatrix}
\rightarrow
\begin{pmatrix}
A_{00} & a_{01} & A_{02} \\
\alpha_{11} & a_{12} & A_{12} \\
A_{20} & a_{21} & A_{22}
\end{pmatrix},
\begin{pmatrix}
p_T \\
p_B
\end{pmatrix}
\rightarrow
\begin{pmatrix}
p_0 \\
p_1 \\
p_2
\end{pmatrix}
\]
| where \(\alpha_{11}\) is \(1 \times 1\), \(\pi_1\) has 1 row |

\[
\begin{pmatrix}
\alpha_{11} \\
a_{21}
\end{pmatrix}, \pi_1 := \text{Pivot}
\begin{pmatrix}
\alpha_{11} \\
a_{21}
\end{pmatrix}
\]

\[
\begin{pmatrix}
a_{10}^T & a_{12}^T \\
A_{20} & A_{22}
\end{pmatrix} := P(\pi_1)
\begin{pmatrix}
a_{10}^T & a_{12}^T \\
A_{20} & A_{22}
\end{pmatrix}
\]

\(a_{21} := a_{21}/\alpha_{11}\)

\(A_{22} := A_{22} - a_{21}a_{12}^T\)

**Continue with**

\(\ldots\)

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

Partition ...
where ...
while \(n(A_{TL}) < n(A)\) do
  Determine block size \(b\)
  Repartition
  \[
  \begin{pmatrix}
  A_{TL} & A_{TR} \\
  A_{BL} & A_{BR}
  \end{pmatrix}
  \rightarrow
  \begin{pmatrix}
  A_{00} & A_{01} & A_{02} \\
  A_{10} & A_{11} & A_{12} \\
  A_{20} & A_{21} & A_{22}
  \end{pmatrix},
  \begin{pmatrix}
  p_T \\
  p_B
  \end{pmatrix}
  \rightarrow
  \begin{pmatrix}
  p_0 \\
  p_1 \\
  p_2
  \end{pmatrix}
  \]
  where \(A_{11}\) is \(b \times b\), \(p_1\) has \(b\) rows

\[\begin{bmatrix}
  \begin{pmatrix}
  A_{11} \\
  A_{21}
  \end{pmatrix}, p_1
  \end{bmatrix} := \text{LUP\_UNB} \left( \begin{pmatrix}
  A_{11} \\
  A_{21}
  \end{pmatrix} \right)\]

\(\begin{pmatrix}
  A_{10} & A_{12} \\
  A_{20} & A_{22}
  \end{pmatrix} := P(p_1) \begin{pmatrix}
  A_{10} & A_{12} \\
  A_{20} & A_{22}
  \end{pmatrix}\)

\(A_{12} := \text{TRILU}(A_{11})^{-1}A_{12}\)
\(A_{22} := A_{22} - A_{21}A_{12}\)

Continue with ...
endwhile
Blocked Algorithm for High Performance

LAPACK implementation: kernels in BLAS

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

,   \( A_{11} \) is \( b \times b \)

1. \( \text{LUP\_UNB} \left( \frac{A_{11}}{A_{21}} \right) \) \quad \text{Unblocked LU, } O(nb^2) \text{ flops}

2. \( A_{12} := \text{TRILU}(A_{11})^{-1}A_{12} \) \quad \text{TRSM, } O(nb^2) \text{ flops}

3. \( A_{22} := A_{22} - A_{21}A_{12} \) \quad \text{GEMM, } O(n^2b) \text{ flops}
Motivation

Basic LU

Practical LU

Parallelization

Control-flow vs. data-flow parallelism

Storage-by-blocks API

New algorithm-by-blocks

Experimental results

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

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

Runtime

1. LUP\_UNB \( \begin{pmatrix} A_{00} \\ A_{10} \\ A_{20} \end{pmatrix} \)

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
Implementation and storage are independent
Matrices stored by blocks are viewed as matrices of matrices
No significative modification to the FLAME codes
1 Motivation
2 Basic LU
3 Practical LU
4 Parallelization
5 New algorithm-by-blocks
   Expose more parallelism
6 Experimental results
7 Concluding remarks
Algorithm-by-blocks for the LU factorization

- Pivoting for stability limits the amount of parallelism

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

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

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

1. Factorize \( P_{11} A_{11} = L_{11} U_{11} \)
2. Apply permutation \( P_{11} \) and factor \( L_{11} \):
   \[
   L_{11}^{-1} P_{11} A_{12} \left| L_{11}^{-1} P_{11} A_{13}
   \right.
   \]
3. Factorize \( P_{21} \left( \frac{A_{11}}{A_{21}} \right) = L_{21} U_{21} \),
4. Apply permutation \( P_{21} \) and factor \( L_{21} \):
   \[
   L_{21}^{-1} P_{21} \left( \frac{A_{12}}{A_{22}} \right) \left| L_{21}^{-1} P_{21} \left( \frac{A_{13}}{A_{23}} \right)
   \right.
   \]
5. Repeat steps 2–4 with \( A_{31} \)
Algorithm-by-blocks for the LU 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 LU factorization with partial 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

1. Motivation
2. Basic LU
3. Practical LU
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 Intel Pentium4 processors</td>
</tr>
</tbody>
</table>

### Implementations

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

LU factorization on SET (16 processors)

- LAPACK dgetrf + multithreaded MKL
- Multithreaded MKL dgetrf
- AB + serial MKL
- AB + serial MKL + contiguous blocks
Experimental Results

LU factorization on NEUMANN (16 processors)

- **LAPACK dgetrf + multithreaded MKL**
- **Multithreaded MKL dgetrf**
- **AB + serial MKL**
- **AB + serial MKL + contiguous blocks**

Matrix size vs. GFLOPS graph showing performance comparison across different methods.
Concluding Remarks

- 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 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
quintana@icc.uji.es

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 $\rightarrow$ irregular problems
  - CellSs $\rightarrow$ 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 + ?