An M-Script API for Parallel Out-of-Core Linear Algebra Operations

Rafael Rubio Enrique S. Quintana-Ortí
{rubio,quintana}@icc.uji.es

High Performance Computing & Architectures Group
Universidad Jaime I de Castellón (Spain)

Motivation

Very large-scale dense linear algebra problems occur in practice!

- Space geodesy (e.g., computation of the gravity field)
- Electromagnetism (as arising from BEM)
- ...

$\longrightarrow$

Problems are too large to fit in-core and use of virtual memory is inefficient

Motivation

As a result, *message-passing* dense linear algebra libraries exist:

- SOLAR
- OOC-ScaLAPACK
- POOCLAPACK

Problems?

- *Message-passing* may not be the best solution for multicore (many-core?)
- Users (scientists and engineers) like high-level environments as MATLAB (or clones SCILAB, OCTAVE), LABVIEW, etc.

Motivation

Our solution:

- Use FLAME@lab, the M-script API to FLAME
- Extend FLASH to OOC
- Extract parallelism from multithreaded BLAS (for now)

Outline

Outline

The Cholesky Factorization

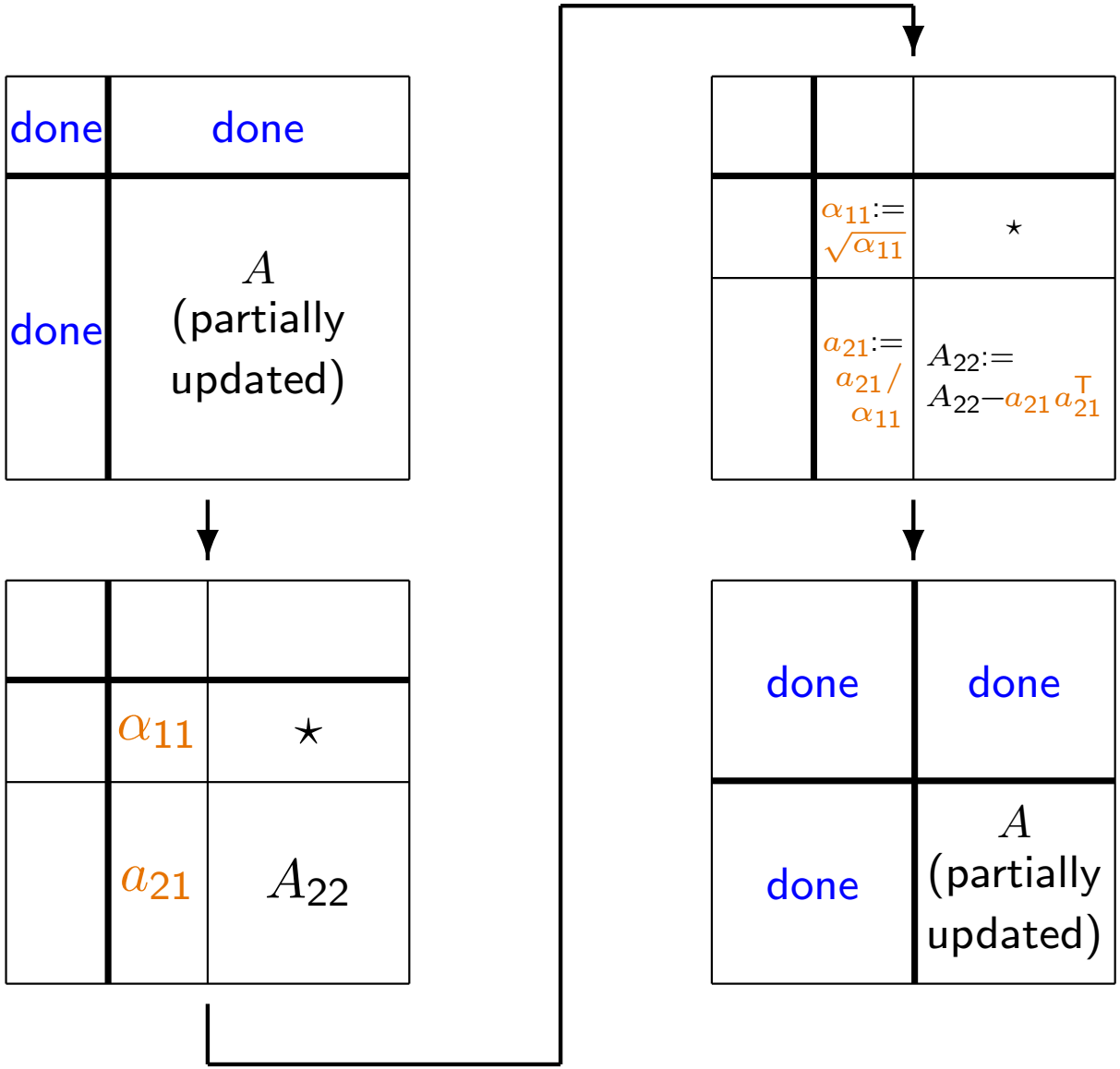
Definition

Given $A \rightarrow n \times n$ symmetric positive definite, compute

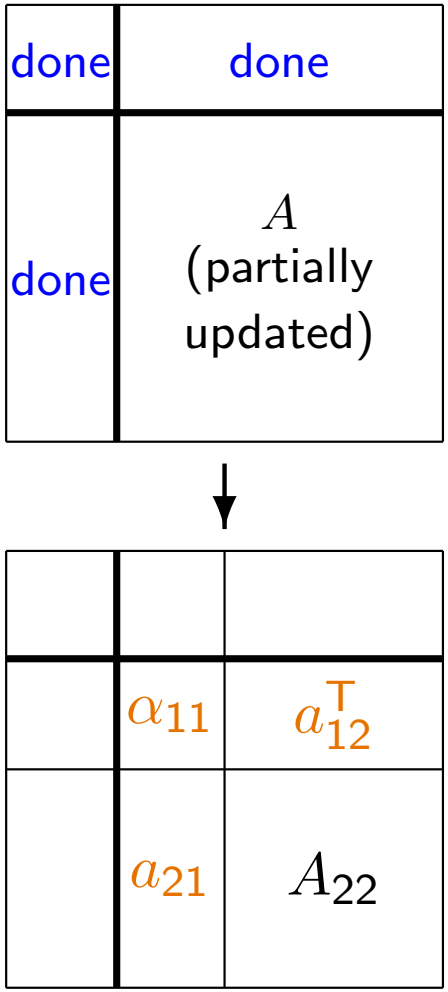
$$A = L \cdot L^T,$$

with $L \rightarrow n \times n$ lower triangular

The Cholesky 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: $[A] := \text{CHOL\_UNB}(A)$

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

Repartition

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

where α_{11} is a scalar

$$\alpha_{11} := \sqrt{\alpha_{11}}$$
$$a_{21} := a_{21}/\alpha_{11}$$
$$A_{22} := A_{22} - a_{21}a_{21}^T$$

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@lab 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@lab code

```
[ A00,  a01,     A02,  ...
  a10t, alpha11, a12t, ...
  A20,  a21,     A22 ] = FLA_Repart_2x2_to_3x3( ATL, ATR, ...
                                                ABL, ABR, ...
                                                1, 1, 'FLA_BR' );
```

(Unblocked) FLAME@lab Code

```
function [ A_out ] = FLA_Cholesky_unb_var1( A )

  %  ... FLA_Part_2x2( ); ...
  while ( size( ABR, 2 ) ~= 0 )
    [ A00,  a01,     A02,  ...
      a10t, alpha11, a12t, ...
      A20,  a21,     A22 ]
         = FLA_Repart_2x2_to_3x3( ATL, ATR, ...
                                  ABL, ABR, ...
                                  1, 1, 'FLA_BR' );
    %-----------------------------------------------%
    alpha11 = sqrt( alpha11 );
    a21     = a21 / alpha11;
    A22     = A22 - tril(a21 * a21');
    %-----------------------------------------------%
    %  ... FLA_Cont_with_3x3_to_2x2( ); ...
  end
  A_out = ATL;

return
```

Blocked FLAME@lab Code

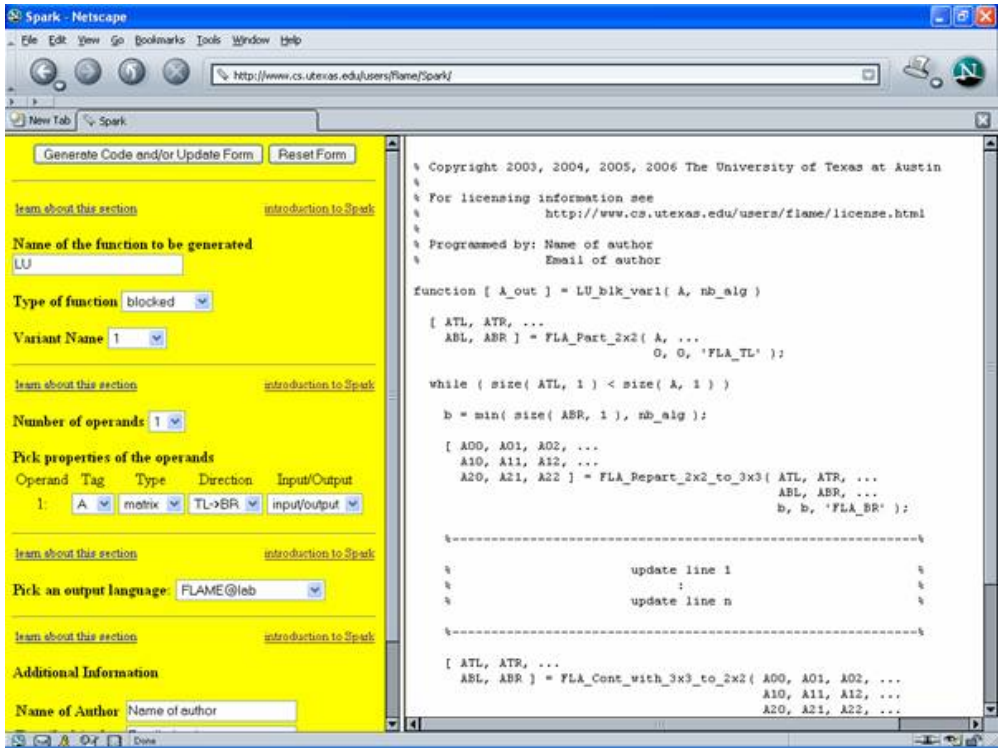
```
function [ A_out ] = FLA_Cholesky_blk_var1( A, nb_alg )

  %  ... FLA_Part_2x2( ); ...
  while ( size( ATL, 2 ) ~= 0 )
    b = min( size( ABR, 1 ), nb_alg );
    [ A00, A01, A02, ...
      A10, A11, A12, ...
      A20, A21, A22 ]
         = FLA_Repart_2x2_to_3x3( ATL, ATR, ...
                                  ABL, ABR, ...
                                  b, b, 'FLA_BR' );
    %-----------------------------------------------%
    FLA_Cholesky_unb( A11 );
    A21 = A21 * inv( tril( A11 ) )';
    A22 = A22 - tril( A21 * A21' );
    %-----------------------------------------------%
    %  ... FLA_Cont_with_3x3_to_2x2( ); ...
  end
  A_out = ATL;

return
```

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

Outline

FLASH

Storage-by-blocks:

- In general, storage-by-blocks enhaces data locality and, therefore, performance...
- at the cost of more complex programming

FLASH

Extending FLAME for hierarchical storage:

- Natural extension: elements of matrices can be themselves matrices (hierarchical/recursive structure)
- FLAME code is independent of actual storage (details hidden in the objects)
- OOC data structures are just one level in this hierarchy, with data stored on disk

OOC API

Driver:

```
% Create matrix
A = FLAOOC_Obj_create( 'FLA_DOUBLE', rows, cols, tile_size,
                       file_name );

% Fill-in: A( i:i+m-1, j:j+n-1 ) = alpha*B+A( i:i+m-1, j:j+n-1 );
% B is m x n
FLAOOC_Axpy_matrix_to_object( alpha, B, A, i, j );

% Cholesky factorization
A = FLAOOC_Cholesky_var2( A );

% Free matrix
FLAOOC_Obj_free( A );
```

OOC to in-core and vice-versa:

FLAOOC_OOC_to_INC

FLAOOC_INC_to_OOC

OOC API

```
function [ A_out ] = FLA_Cholesky_blk_var2( A, nb_alg )

  %  ... FLA_Part_2x2( ); ...
  while ( size( ABR, 2 ) ~= 0 )
    b = min( size( ABR, 1 ), nb_alg );
    [ A00, A01, A02, ...
      A10, A11, A12, ...
      A20, A21, A22 ]
         = FLA_Repart_2x2_to_3x3( ATL, ATR, ...
                                  ABL, ABR, ...
                                  b, b, 'FLA_BR' );
    %-----------------------------------------------%
    A10 = A10 \ tril( A00 )';
    A11 = A11 - tril( A10 * A10' );
    A11 = FLA_Cholesky_unb( A11 );
    %-----------------------------------------------%
    %  ... FLA_Cont_with_3x3_to_2x2( ); ...
  end
  A_out = ATL;

return
```

OOC API

```
function [ A_out ] = FLAOOC_Cholesky_var2( A )

  %  ... FLAOOC_Part_2x2( ); ...
  while ( FLAOOC_Obj_width( ABR ) ~= 0 )
    %  ... FLAOOC_Repart_2x2_to_3x3( );
    %-----------------------------------------------%
    A10  = FLAOOC_Trsm( % Mode arguments
                        1, A00,
                           A10 );   % A10 := A10 * TRIL(A00)^-T
    AINC = FLAOOC_OOC_to_INC( A11 );% Copy A11 in-core
    AINC = FLAOOC_Syrk( % Mode arguments
                        -1, A10,
                         1, AINC ); % A11 := A11 - A10 * A10^T
    AINC = chol(AINC)';             % A11 := chol( A11 )
    FLAOOC_INC_to_OOC( AINC, A11 ); % Store A11 out-of-core
    %-----------------------------------------------%
    %  ... FLAOOC_Cont_with_3x3_to_2x2( ); ...
  end
  A_out = ATL;

return
```

Parallelization on Multithreaded Architectures

Traditional approach

- Link with multithreaded BLAS

Advanced approach (future)

- Extract parallelism at higher level: SuperMatrix runtime system

Outline

Experimental Results

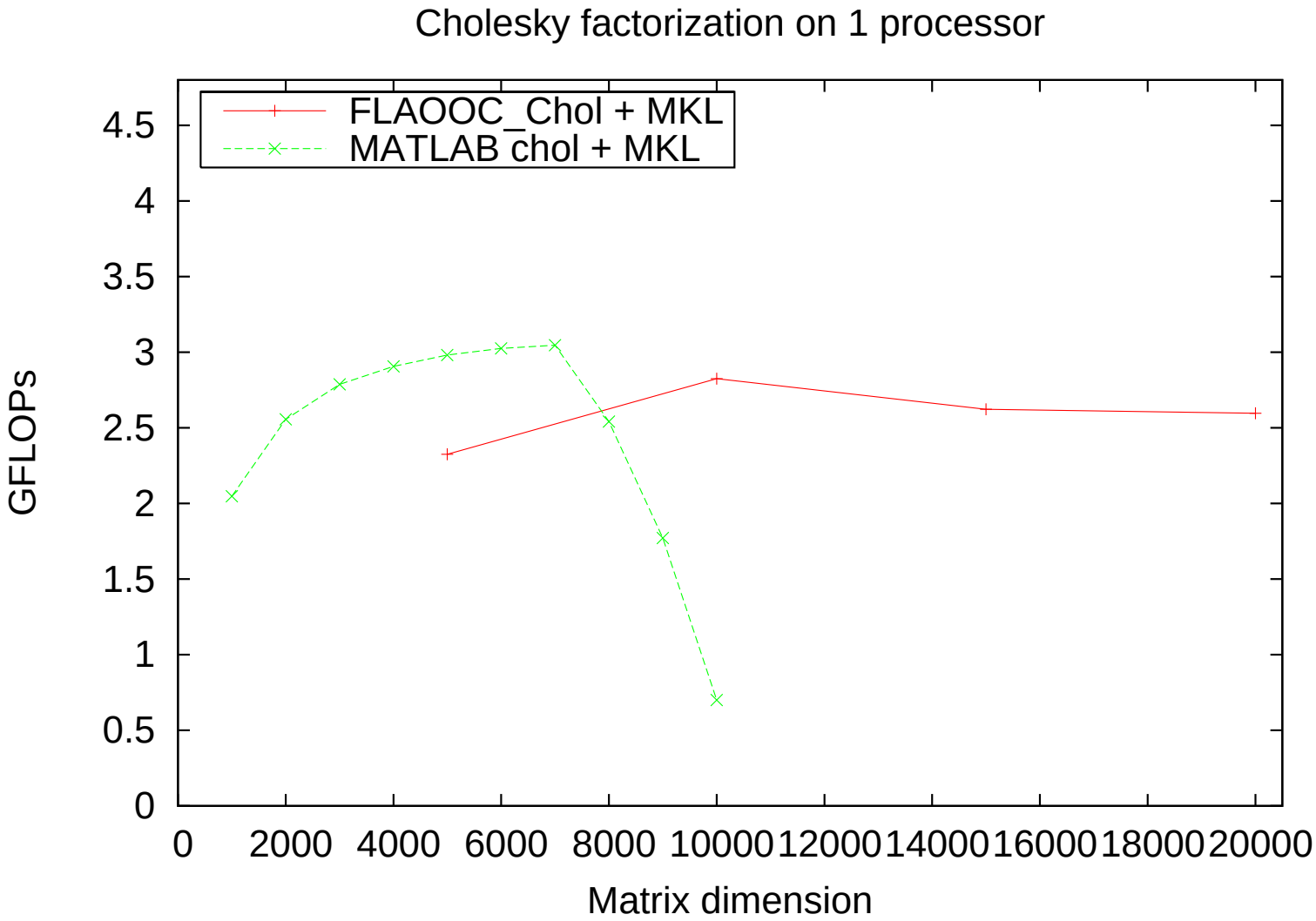
General

Platform	Specs.
MATSERV	SMP with two Intel Xeon@2.4 GHz and 1 GB of RAM
CAT	SMP with four Intel Itanium2@1.5 GHz and 4 GB of RAM

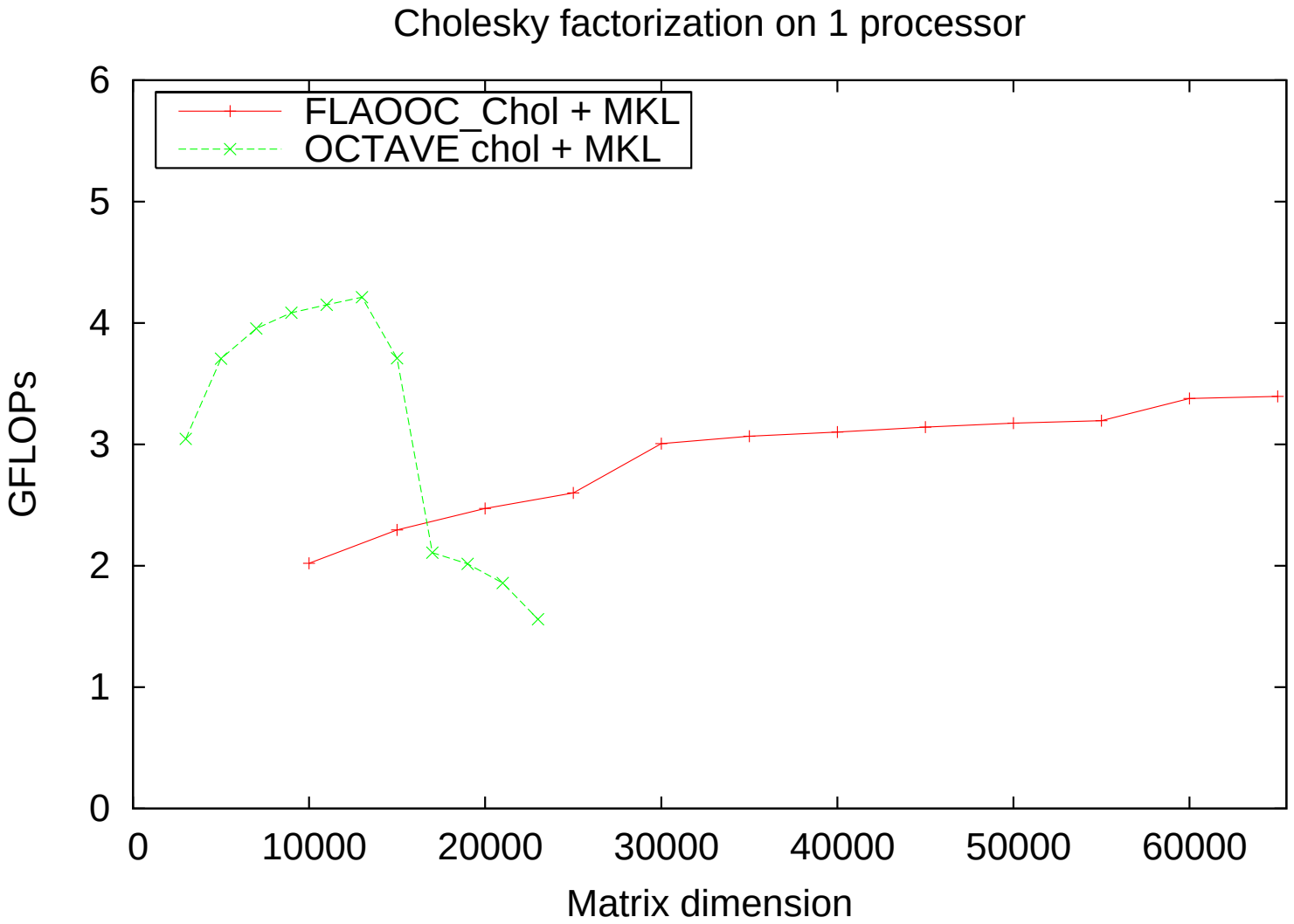
Implementations

- MATLAB/OCTAVE routine `chol` linked to multithreaded MKL (7.0 or higher)
- `FLA_Cholesky_blk` linked to multithreaded MKL

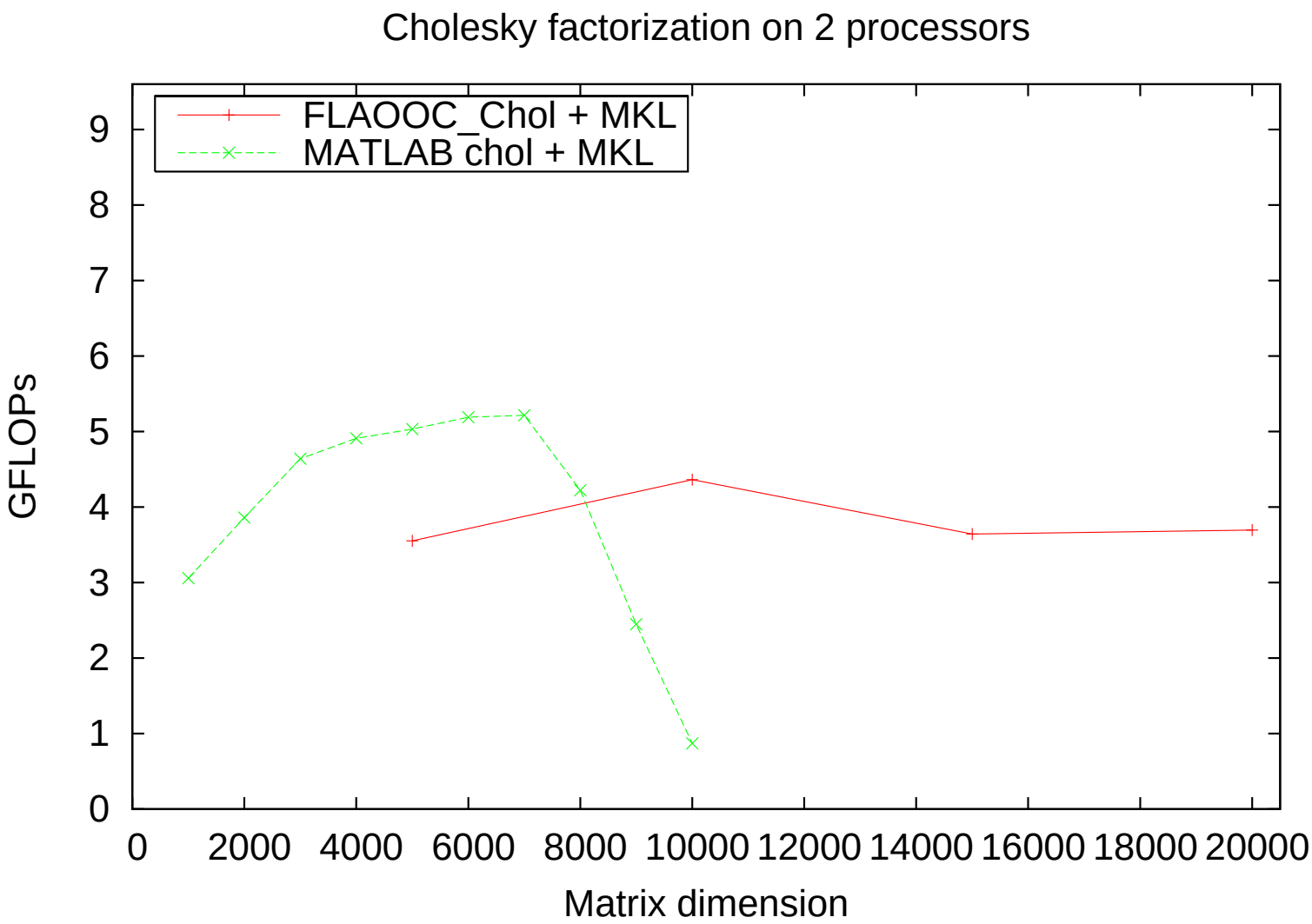
Experimental Results: MATSERV



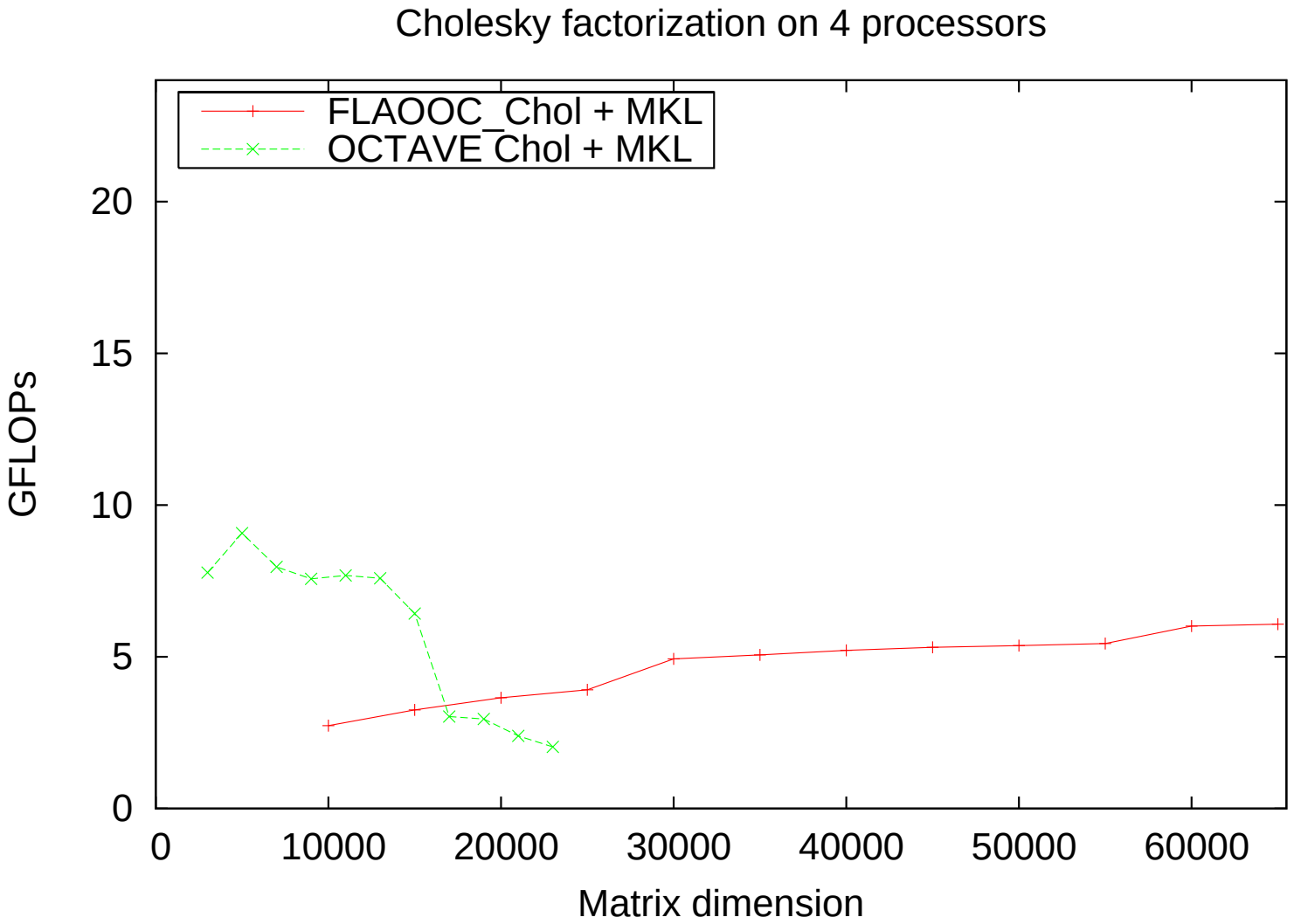
Experimental Results: CAT



Experimental Results: MATSERV



Experimental Results: CAT



Concluding Remarks

- FLAOOC@lab is an easy-to-use tool to develop OOC codes for dense linear algebra operations
- The new API allows the solution of large linear systems from MATLAB/OCTAVE
- Focus is on programmability but performance will come with a similar C API

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