UNIVERSITAT JAUME·I
HPC&A

Easy Programming of Linear Algebra Operations on Hybrid CPU-GPU Platforms

Enrique S. Quintana-Ortí

Index

UNIVERSITAT JAUME·I
HPC&A

- The libflame library

- The StarSs framework

UNIVERSITAT JAUME·I

GPU support

Disclaimer

UNIVERSITAT JAUME·I
HPC&A

- Not a course on low-level programming of dense linear algebra kernels on GPUs!
  - V. Volkov, J. Demmel. "Benchmarking GPUs to tune dense linear algebra", SC08
  - L-S. Chien, "Hand-tuned SGEMM on GT200 GPU", TR Tsing Hua Univ., Taiwan

+

- Sorry if the performance numbers of some "products" do not look so good…

Initial considerations

UNIVERSITAT JAUME·I
HPC&A

- No low level programming: assume there is a tuned BLAS for GPU/general-purpose cores
- High level approach: minimal changes to existing codes/solutions
  - Abstract programmer from data transfers, without loosing performance
- Runtime-based solution for multi-GPU platforms

Index

UNIVERSITAT JAUME·I
HPC&A

- The libflame library
  1. A user's view
  2. Creating your own algorithm
  3. FLAME runtime
  4. Clusters of GPUs
- The StarSs framework

Index

UNIVERSITAT JAUME·I
HPC&A

- The libflame library

1. A user's view	1. Introduction
2. Creating your own algorithm	2. Configuration
3. FLAME runtime	3. Operation status
4. Clusters of GPUs	4. Examples

- The StarSs framework

What is libflame?

- A library?
- A framework?
- A repository of algorithmic variants?

Who are the targets?

UNIVERSITAT JAUME·I
HPC&A

- Newcomers
- Current users of BLAS, LAPACK
- Some PLAPACK/ScaLAPACK users whose problems are too small

Current Coverage

- Level-1, -2, and -3 BLAS
  - No banded support
- Basic LAPACK operations
  - No SVD/EVD support (yet)
- Operations not implemented by LAPACK
  - Example: up-and-downdating
- Supporting utility functions

Key Features

- Formally derived algorithms
- Object-based API (abstraction!)
- High performance
- Storage: column-, row-major, AND general
  - Or a mixture!
- Dependency-aware multithreading
- Build system(s)
  - GNU and Windows

Where is libflame available?

- FLAME website
  - www.cs.utexas.edu/users/flame/libflame/
  - Milestone releases
  - Nightly snapshots
  - Available as free software under LGPL

libflame for GNU/Linux

UNIVERSITAT JAUME·I
HPC&A

- Software requirements
  - C compiler
  - GNU bash (2.0 or later)
  - GNU make
  - BLAS library
  - OpenMP-aware C compiler (optional)

- Build system
  - `./configure; make; make install`

libflame for Windows

UNIVERSITAT JAUME·I
HPC&A

- Software requirements
  - C compiler
  - Python (2.6 or later)
  - Microsoft nmake
  - BLAS library
  - OpenMP-aware C compiler (optional)

- Build system
  - `.\configure.cmd; nmake; nmake install`

UNIVERSITAT JAUME·I
HPC&A

Options

- Static/dynamic library generation
- lapack2flame compatibility layer
- Multithreading: POSIX threads or OpenMP
- GPU support
- Memory alignment
- Internal error checking

A sampling of functionality

UNIVERSITAT JAUME·I
HPC&A

operation	Classic FLAME	FLASH/SM	lapack2flame
Level-3 BLAS	y	y	n/a
Cholesky	y	y	y
LU with partial pivoting	y	y	y
LU with incremental pivoting	y	y	*
QR (UT)	y	y	y
LQ (UT)	y	y	y
SPD/HPD inversion	y	y	y
Triangular inversion	y	y	y
Triangular Sylvester	y	y	y
Lyapunov	y	y	y
Up-and-downdate (UT)	y	y	*
SVD	planned		
EVD	planned		

* Not present in LAPACK

Types of interfaces

UNIVERSITAT JAUME·I
HPC&A

- Classic
  - Matrix type: flat
  - Task-level parallelism: no
    - Suitable for multithreaded BLAS
- FLASH
  - Matrix type: hierarchical
  - Task-level parallelism: SuperMatrix
    - Sequential execution is optional

External buffers

UNIVERSITAT JAUME·I
HPC&A

```
FLA_Obj A;

// Initialize conventional matrix: buffer, m, rs, cs


FLA_Init();

FLA_Obj_create_without_buffer( FLA_DOUBLE, m, m, &A );
FLA_Obj_attach_buffer( buffer, rs, cs, &A );


FLA_Chol( FLA_LOWER_TRIANGULAR, A );

FLA_Obj_free_without_buffer( &A );

FLA_Finalize();
```

Native objects

```
FLA_Obj A;

// Initialize conventional matrix: buffer, m, rs, cs


FLA_Init();

FLA_Obj_create( FLA_DOUBLE, m, m, 0, 0, &A );
FLA_Copy_buffer_to_object( FLA_NO_TRANSPOSE, m, m,
                           buffer, rs, cs, 0, 0, A );

FLA_Chol( FLA_LOWER_TRIANGULAR, A );

FLA_Obj_free( &A );

FLA_Finalize();
```

Storage-by-blocks (seq)

UNIVERSITAT JAUME·I HPC&A

```
FLA_Obj A;

// Initialize conventional matrix: buffer, m, rs, cs
// Obtain storage blocksize b.

FLA_Init();

FLASH_Obj_create( FLA_DOUBLE, m, m, 1, &b, &A );
FLASH_Copy_buffer_to_hier( m, m, buffer, rs, cs,
                           0, 0, A );
FLASH_Queue_disable();
FLASH_Chol( FLA_LOWER_TRIANGULAR, A );

FLASH_Obj_free( &A );

FLA_Finalize();
```

Storage-by-blocks (MT)

UNIVERSITAT JAUME·I HPC&A

```
FLA_Obj A;

// Initialize conventional matrix: buffer, m, rs, cs
// Obtain storage blocksize, # of threads: b, n_threads

FLA_Init();

FLASH_Obj_create( FLA_DOUBLE, m, m, 1, &b, &A );
FLASH_Copy_buffer_to_hier( m, m, buffer, rs, cs,
                           0, 0, A );
FLASH_Queue_set_num_threads( n_threads );
FLASH_Chol( FLA_LOWER_TRIANGULAR, A );

FLASH_Obj_free( &A );

FLA_Finalize();
```

Solving a system via LU

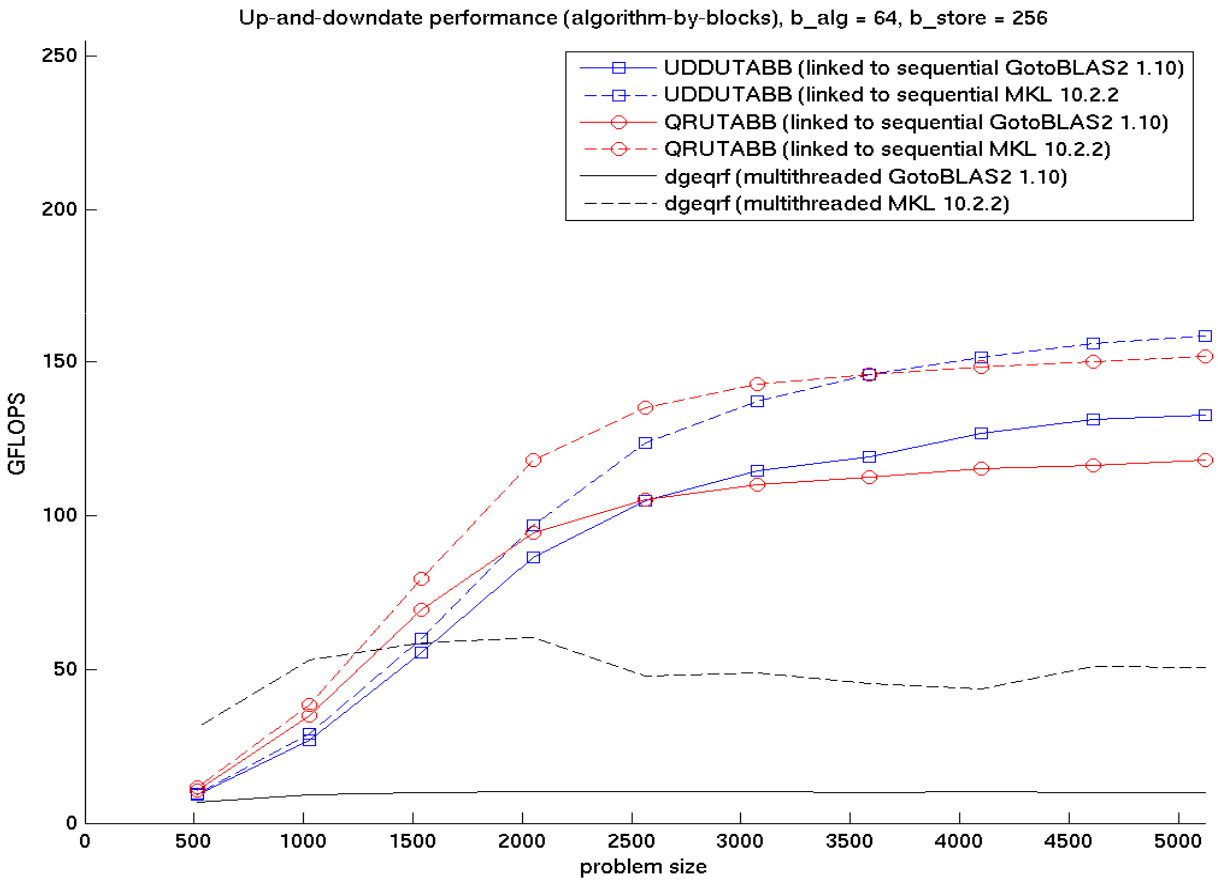
UNIVERSITAT JAUME·I
HPC&A

```
FLA_Obj A, p, b, x;

FLA_Obj_create( FLA_DOUBLE, m, m, 0, 0, &A );
FLA_Obj_create( FLA_DOUBLE, m, 1, 0, 0, &b );
FLA_Obj_create( FLA_DOUBLE, m, 1, 0, 0, &x );
FLA_Obj_create( FLA_INT,    m, 1, 0, 0, &p );

// Initialize A, b, x.

FLA_LU_piv( A, p );
FLA_LU_piv_solve( A, p, b, x );

FLA_Obj_free( &A );
FLA_Obj_free( &b );
FLA_Obj_free( &x );
FLA_Obj_free( &p );
```

Solving a system via QR


```
FLA_Obj A, T, b, x;

FLA_Obj_create( FLA_DOUBLE, m, n, 0, 0, &A );
FLA_Obj_create( FLA_DOUBLE, m, 1, 0, 0, &b );
FLA_Obj_create( FLA_DOUBLE, n, 1, 0, 0, &x );

// Initialize A, b, x.

FLA_QR_UT_create_T( A, &T );

FLA_QR_UT( A, T );
FLA_QR_UT_solve( A, T, b, x );

FLA_Obj_free( &A );
FLA_Obj_free( &b );
FLA_Obj_free( &x );
```

Up-and-Downdate



Index

UNIVERSITAT JAUME·I
HPC&A

- The libflame library

1. A user's view
2. Creating your own algorithm
3. FLAME runtime
4. Clusters of GPUs

→

1. FLAME notation and algorithms
2. Spark: from algorithm to code
3. Running on multicore

- The StarSs framework

The Cholesky factorization

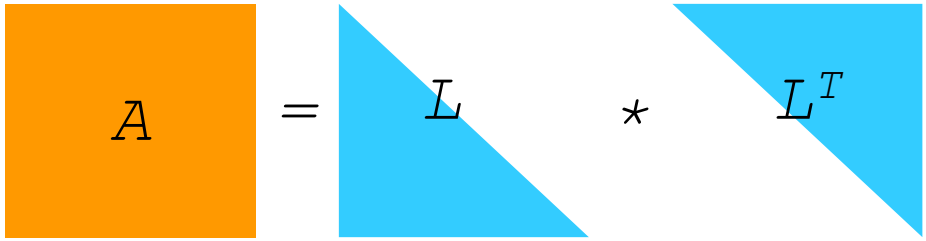
UNIVERSITAT JAUME·I
HPC&A

```
FLA_Obj A;

// Initialize conventional matrix: buffer, m, rs, cs


FLA_Init();

FLA_Obj_create_without_buffer( FLA_DOUBLE, m, m, &A );
FLA_Obj_attach_buffer( buffer, rs, cs, &A );

FLA_Chol( FLA_LOWER_TRIANGULAR, A );

FLA_Obj_free_without_buffer( &A );

FLA_Finalize();
```

The Cholesky factorization

UNIVERSITAT JAUME·I HPC&A

- Lower triangular case:



Key in the solution of s.p.d. linear systems

$$A\ x = b \equiv (LL^T)\ x = b$$

$$L\ y = b \Rightarrow y$$

$$L^T\ x = y \Rightarrow x$$

Algorithm

$$A \rightarrow \begin{pmatrix} \alpha_{11} & * \\ a_{21} & A_{22} \end{pmatrix} \qquad L \rightarrow \begin{pmatrix} \lambda_{11} & \\ l_{21} & L_{22} \end{pmatrix}$$

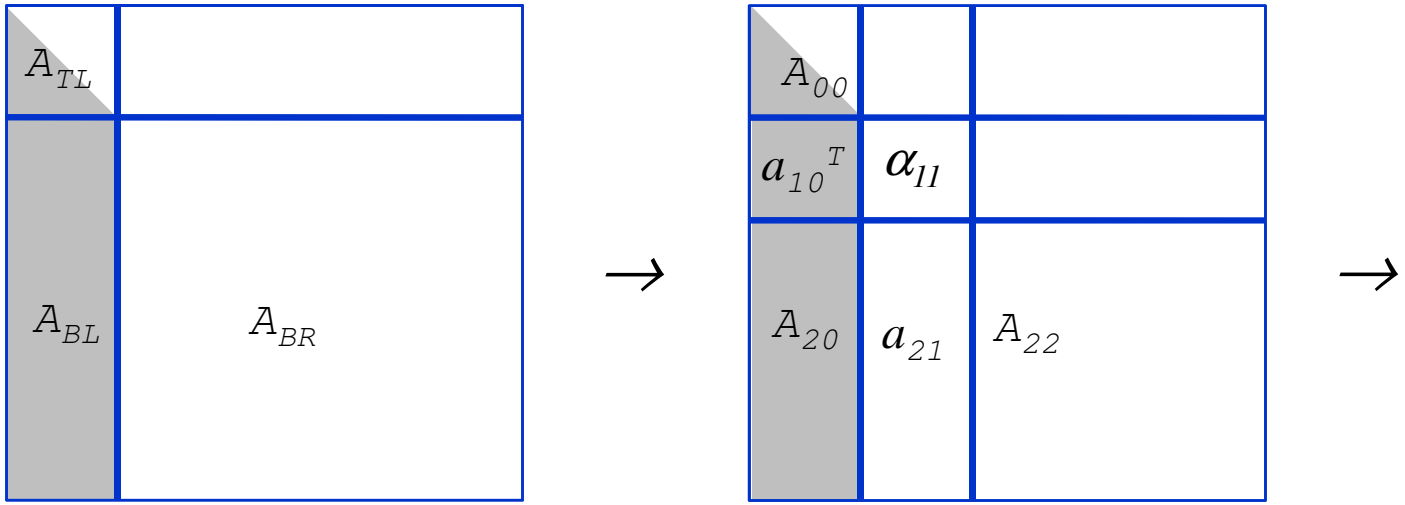
$$A = L * L^T \equiv \begin{pmatrix} \alpha_{11} & * \\ a_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} \lambda_{11} & \\ l_{21} & L_{22} \end{pmatrix} * \begin{pmatrix} \lambda_{11} & \\ l_{21} & L_{22} \end{pmatrix}^T \rightarrow$$

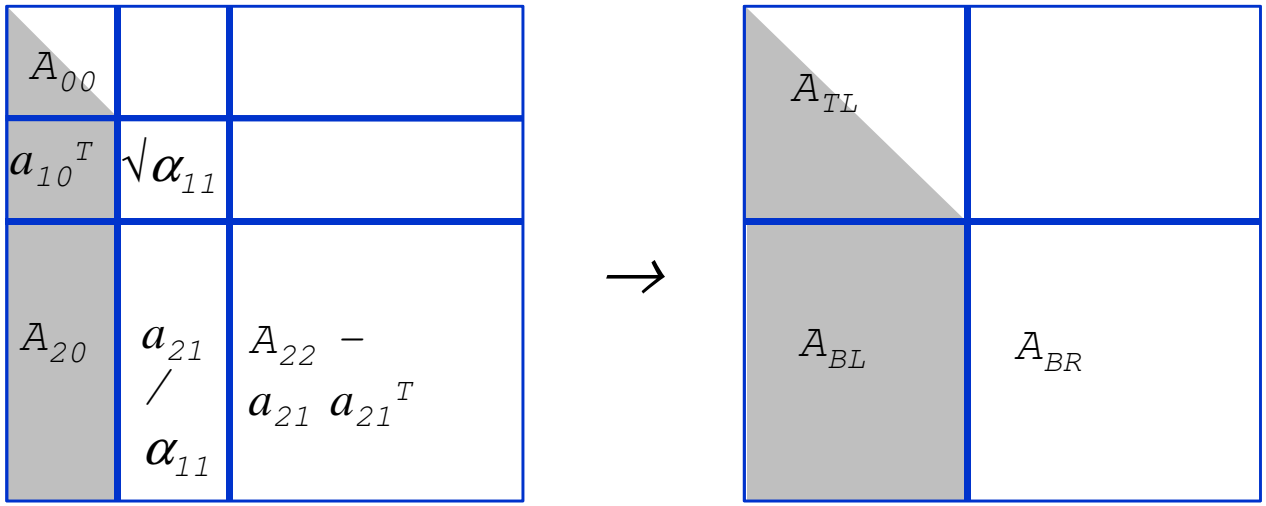
$$\alpha_{11} = \lambda_{11}\lambda_{11}^T \rightarrow \lambda_{11} = \sqrt{\alpha_{11}}$$

$$a_{21} = l_{21}\ \lambda_{11} \rightarrow l_{21} = a_{21}/\lambda_{11}$$

$$A_{22} = L_{22}L_{22}^T + a_{21}\,a_{21}^T \rightarrow (A_{22} - a_{21}\,a_{21}^T) = L_{22}L_{22}^T$$

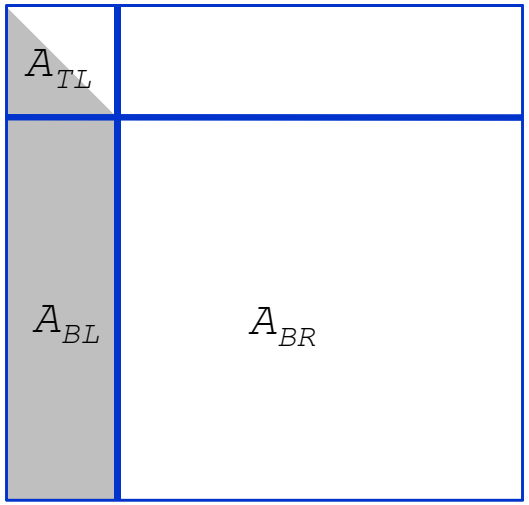
Algorithm loop: repartition+operation+merging

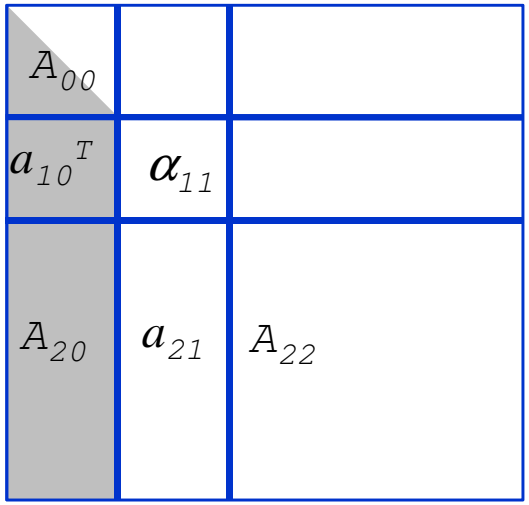




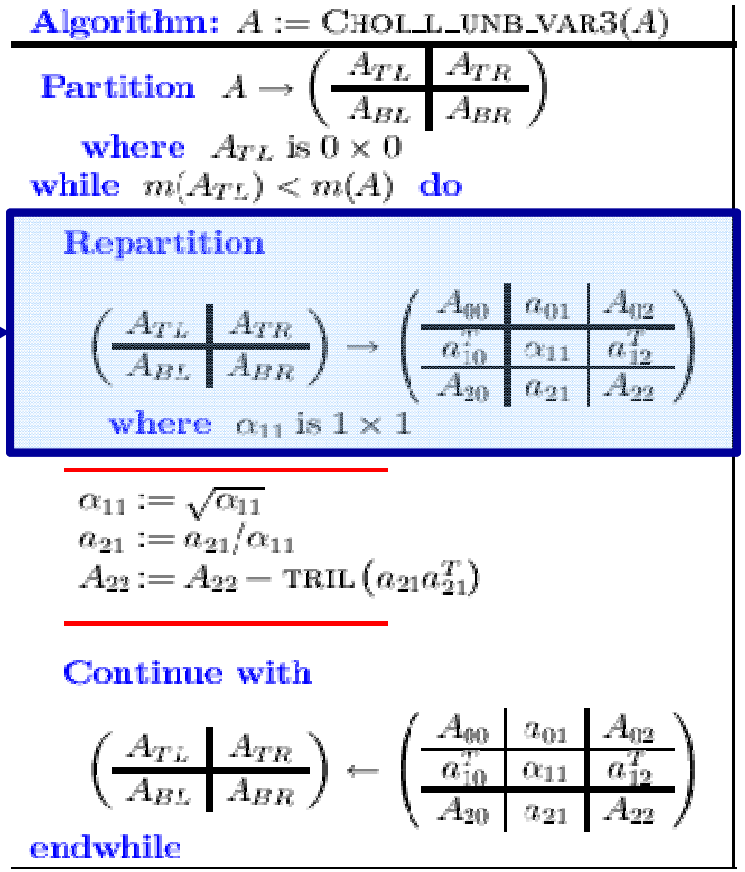
UNIVERSITAT JAUME·I
HPC&A

Algorithm loop: repartition





→



Indexing operations

Algorithm loop: operation

A_{00}		
a_{10}^T	α_{11}	
A_{20}	a_{21}	

→

A_{00}		
a_{10}^T	$\sqrt{\alpha_{11}}$	
A_{20}	a_{21} / α_{11}	$A_{22} - a_{21}\ a_{21}^T$

Algorithm: $A := \text{CHOL\_L\_UNB\_VAR3}(A)$

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 $m(A_{TL}) < m(A)$ **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 1×1

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

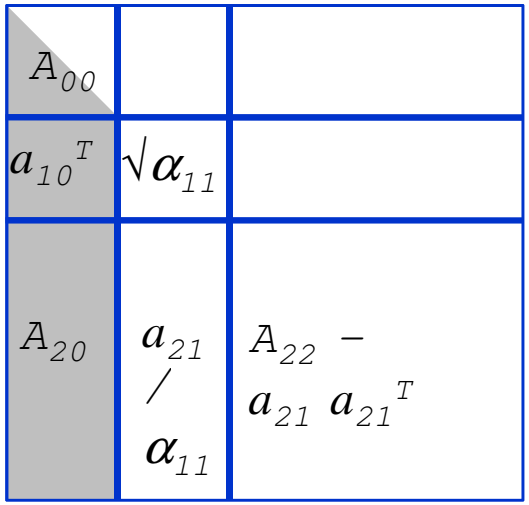
Continue with

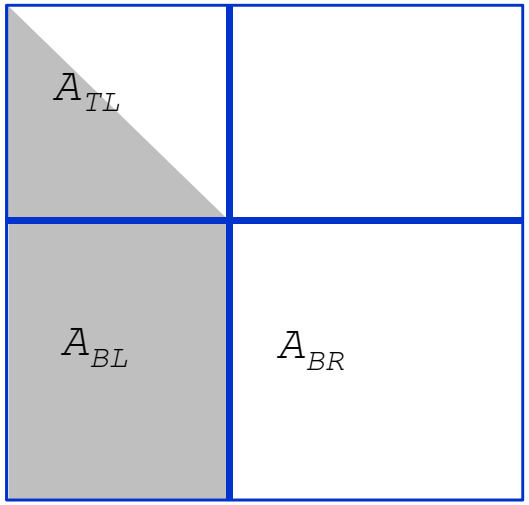
$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$$

endwhile

Real computation

Algorithm loop: merging





→

Algorithm: $A := \text{CHOL\_L\_UNB\_VAR3}(A)$

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 $m(A_{TL}) < m(A)$ **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 1×1

$\alpha_{11} := \sqrt{\alpha_{11}}$
$a_{21} := a_{21}/\alpha_{11}$
$A_{22} := A_{22} - \text{TRIL}\left(a_{21}a_{21}^T\right)$

Continue with

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

endwhile

Indexing operation

Algorithm

- Automatic development from math. specification

$$A = L * L^T$$

Mechanical procedure

Algorithm: $A := \text{CHOL\_L\_UNB\_VAR3}(A)$

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 $m(A_{TL}) < m(A)$ **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 1×1

$$\alpha_{11} := \sqrt{\alpha_{11}}$$
$$a_{21} := a_{21}/\alpha_{11}$$
$$A_{22} := A_{22} - \text{TRIL}\left(a_{21}a_{21}^T\right)$$

Continue with

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

endwhile

APIs

Algorithm: $A := \text{CHOL\_L\_UNB\_VAR3}(A)$

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 $m(A_{TL}) < m(A)$ **do**

Repartition

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

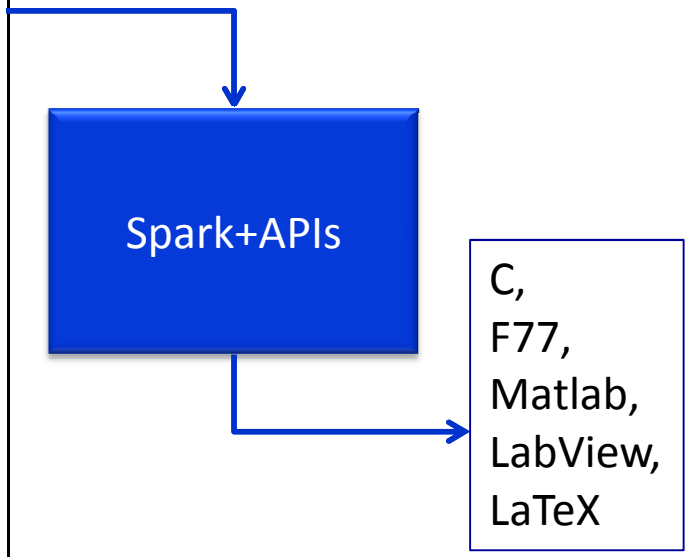
where α_{11} is 1×1

$$\begin{array}{l} \alpha_{11} := \sqrt{\alpha_{11}} \\ a_{21} := a_{21}/\alpha_{11} \\ A_{22} := A_{22} - \text{TRIL}\left(a_{21}a_{21}^T\right) \end{array}$$

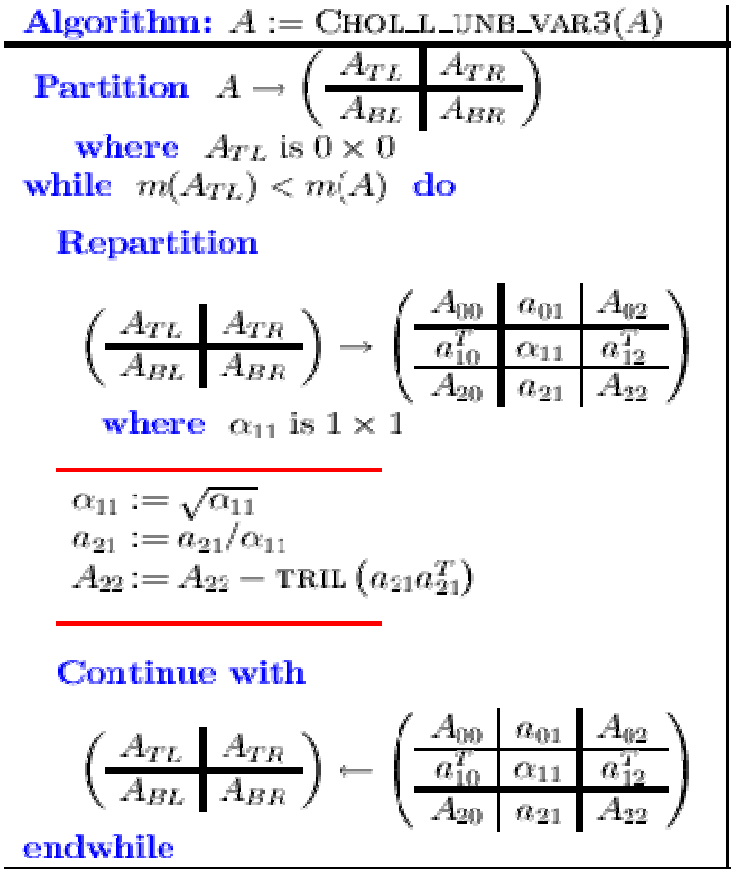
Continue with

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

endwhile

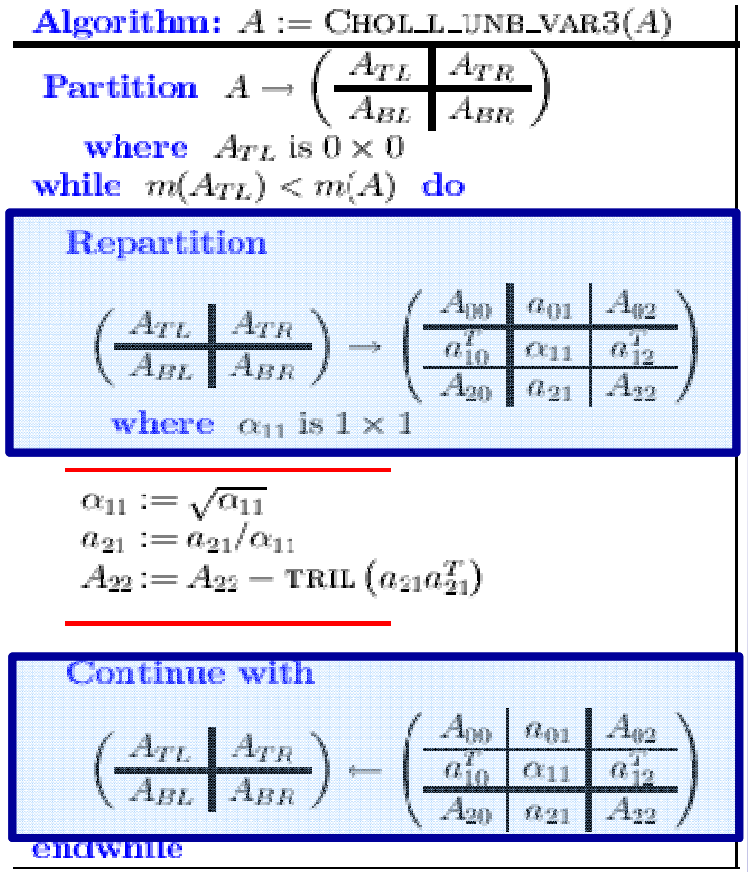


Spark website



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

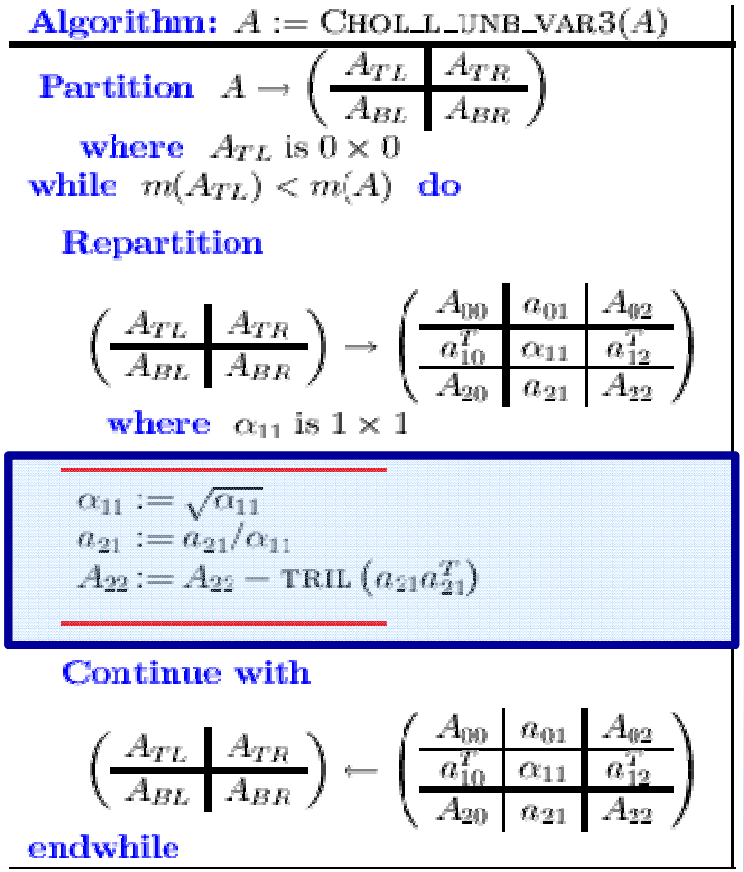
Example: FLAME@lab




```
[ ATL, ATR,...
  ABL, ABR ] =
  FLA_Part_2x2( A, 0, 0, 'FLA_TL' );

while ( size( ATL, 1 ) < size( A, 1 ) )

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

  % :

  %-------------------------------------%
  [ ATL, ATR,...
    ABL, ABR ] = ...
  FLA_Cont_with_3x3_to_2x2( A00, a01, A02,...
                            a10t, alpha11, a12t,...
                            A20, a21, A22,...
                            'FLA_TL' );
end
```

Indexing operations

Example: FLAME@lab

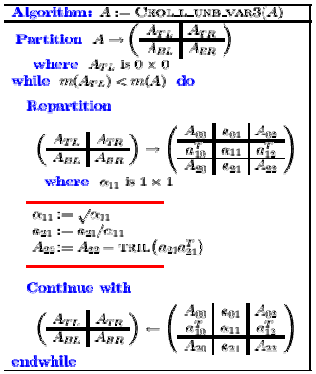


- Manually fill-in operations

```
[…] = FLA_Part_2x2(…);

while ( size( ATL, 1 ) < size( A, 1 ) )

  […] = FLA_Repart_2x2_to_3x3(…);

  %-------------------------------------%
  alpha11 = sqrt( alpha11 );
  a21     = a21 / alpha11;
  A22     = A22 - tril( a21*a21' );
  %-------------------------------------%

  […] = FLA_Cont_with_3x3_to_2x2(…);


end
```

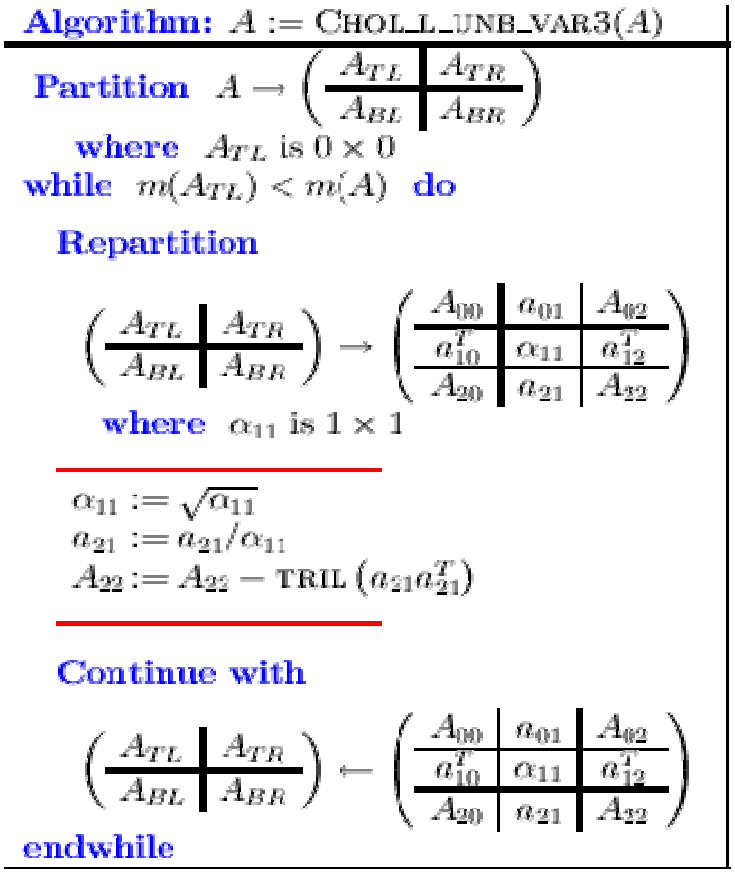
Real computation

Example: FLAMEC




```
FLA_Part_2x2( A,  &ATL,  &ATR,
                  &ABL,  &ABR,  0,  0,  FLA_TL );

while ( FLA_Obj_length( ATL ) < FLA_Obj_length( A ) ){

  b = min( FLA_Obj_length( ABR ), nb_alg );

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

  FLA_Cont_with_3x3_to_2x2( &ATL, /**/ &ATR,      A00,  a01,     /**/ A02,
                                                  a10t, alpha11, /**/ a12t,
                         /* ************** */ /* ************************/
                            &ABL, /**/ &ABR,      A20,  a21,     /**/ A22,
                            FLA_TL );

}
```

Example: FLAMEC



Algorithm: $A := \text{CHOL\_L\_UNB\_VAR3}(A)$

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 $m(A_{TL}) < m(A)$ **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 1×1

$\alpha_{11} := \sqrt{\alpha_{11}}$

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

$A_{22} := A_{22} - \text{TRIL}\left(a_{21}a_{21}^T\right)$

Continue with

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

endwhile

- libflame employs external BLAS: GotoBLAS, MKL, ACML, ATLAS, netlib

```
FLA_Part_2x2(…);

while ( FLA_Obj_length(ATL) < FLA_Obj_length(A) ){

 FLA_Repart_2x2_to_3x3(…);

  /*--------------------------------------*/
  FLA_Sqrt( alpha11 );
  FLA_Inv_scal( alpha11, a21 );
  FLA_Syr( FLA_LOWER_TRIANGULAR, FLA_NO_TRANSPOSE,
           FLA_MINUS_ONE, a21, A22 );
  /*--------------------------------------*/

  FLA_Cont_with_3x3_to_2x2(…);

}
```

Index

UNIVERSITAT JAUME·I
HPC&A

- The libflame library
  1. A user's view
  2. Creating your own algorithm
  3. FLAME runtime
  4. Cluster of GPUs

  → 1. Task parallelism
    2. SuperMatrix
    3. GPU support
- The SMPs/GPUSs framework

Data-flow parallelism? Dynamic scheduling? Run-time?

UNIVERSITAT JAUME·I
HPC&A

- Surely not a new idea…
  - Cilk
  - StarSs (GridSs)
  - StarPU
  - …

- "An Efficient Algorithm for Exploiting Multiple Arithmetic Units", R. M. Tomasulo, IBM J. of R&D, Volume 11, Number 1, Page 25 (1967)

The basis for exploitation of ILP
on current superscalar processors!

The TEXT project

UNIVERSITAT JAUME·I HPC&A

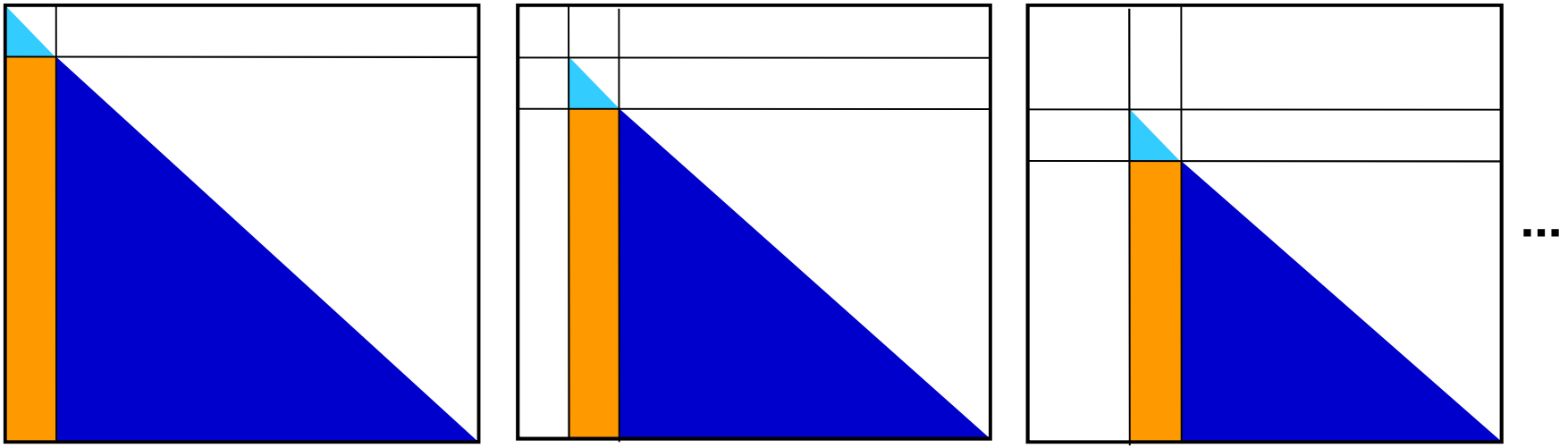
- Towards Exaflop applicaTions

- Demonstrate that **Hybrid MPI/SMPSs** addresses the Exascale challenges in a productive and efficient way.
  - Deploy at supercomputing centers: Julich, EPCC, HLRS, BSC
  - Port Applications (HLA, SPECFEM3D, PEPC, PSC, BEST, CPMD, LS1 MarDyn) and develop algorithms.
  - Develop additional environment capabilities
    - tools (debug, performance)
    - improvements in runtime systems (load balance and GPUSs)
  - Support other users
    - Identify users of TEXT applications
    - Identify and support interested application developers
  - Contribute to Standards (OpenMP ARB, PERI-XML)

Blocked algorithms

- Cholesky factorization

$$A_{11} = L_{11}\, L_{11}^{T}$$
$$A_{21} := L_{21} = A_{21}\, L_{11}^{-T}$$
$$A_{22} := A_{22} - L_{21}\, L_{21}^{T}$$



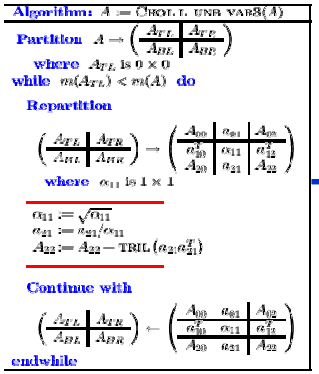
1st iteration 2nd iteration 3rd iteration

Blocked algorithms

UNIVERSITAT JAUME·I HPC&A

- Cholesky factorization

$$A = L * L^T$$



APIs + Tools

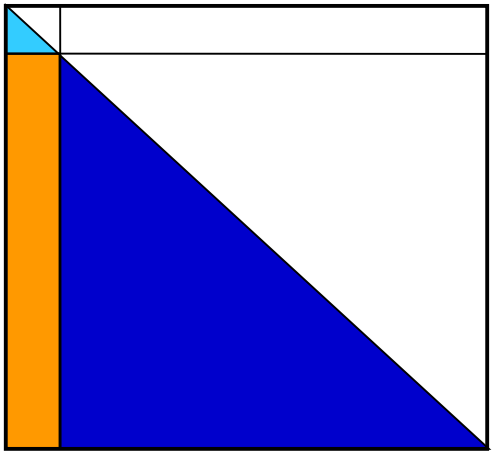
```
FLA_Part_2x2(…);

while ( FLA_Obj_length(ATL) < FLA_Obj_length(A) ){

 FLA_Repart_2x2_to_3x3(…);

  /*--------------------------------------*/
  FLA_Chol( FLA_LOWER_TRIANGULAR, A11 );
  FLA_Trsm( FLA_RIGHT, FLA_LOWER_TRIANGULAR,
            FLA_TRANSPOSE, FLA_NONUNIT_DIAG,
            FLA_ONE, A11, A21 );
  FLA_Syrk( FLA_LOWER_TRIANGULAR,FLA_NO_TRANSPOSE,
            FLA_MINUS_ONE, A21, FLA_ONE, A22 );
  /*--------------------------------------*/

  FLA_Cont_with_3x3_to_2x2(…);

}
```

Blocked algorithms

UNIVERSITAT JAUME·I HPC&A

- Simple parallelization: link with MT BLAS

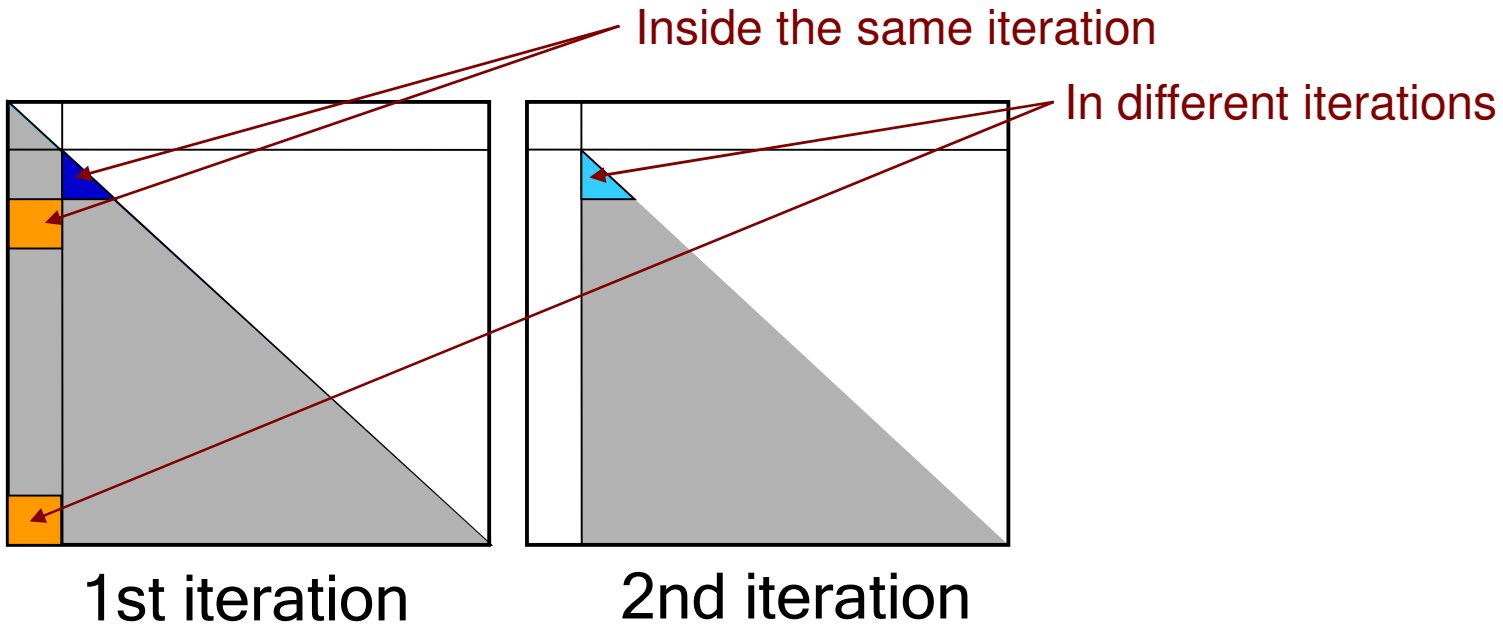
$A_{11} = L_{11} L_{11}^T$
$A_{21} := L_{21} = A_{21} L_{11}^{-T}$
$A_{22} := A_{22} - L_{21} L_{21}^T$



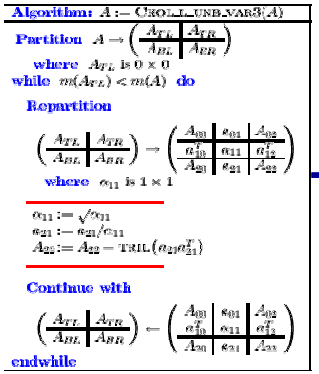
```
FLA_Part_2x2(…);

while ( FLA_Obj_length(ATL) < FLA_Obj_length(A) ){

 FLA_Repart_2x2_to_3x3(…);

  /*-------------------------------------*/
  FLA_Chol( FLA_LOWER_TRIANGULAR, A11 );
  FLA_Trsm( FLA_RIGHT, FLA_LOWER_TRIANGULAR,
            FLA_TRANSPOSE, FLA_NONUNIT_DIAG,
            FLA_ONE, A11, A21 );
  FLA_Syrk( FLA_LOWER_TRIANGULAR,FLA_NO_TRANSPOSE,
            FLA_MINUS_ONE, A21, FLA_ONE, A22 );
  /*-------------------------------------*/

  FLA_Cont_with_3x3_to_2x2(…);

}
```

Blocked algorithms

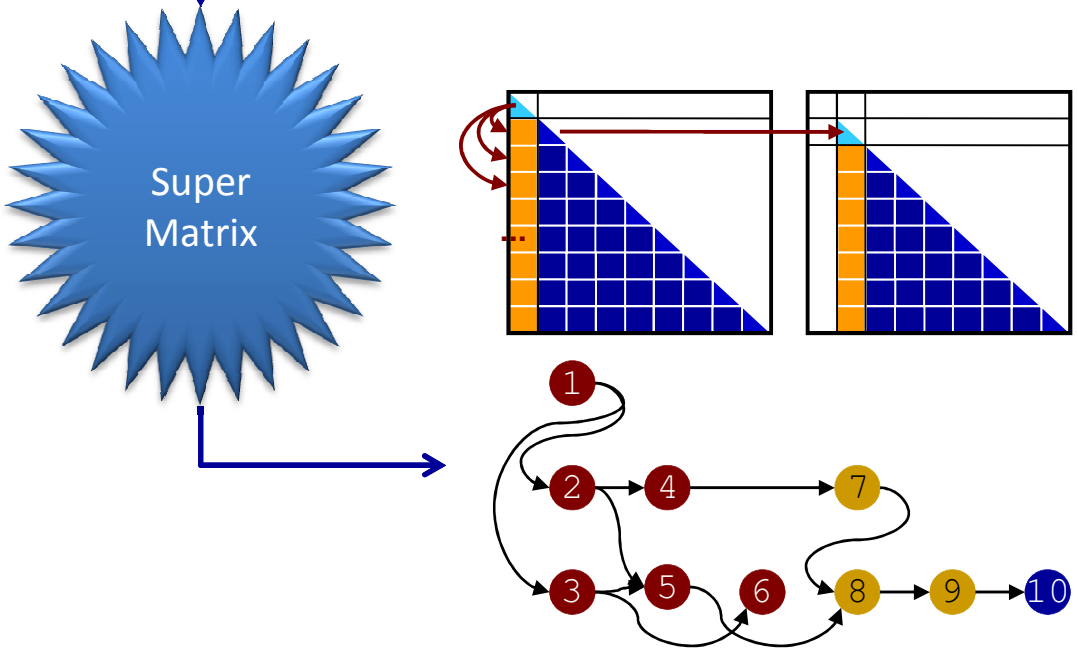
- There is more parallelism!



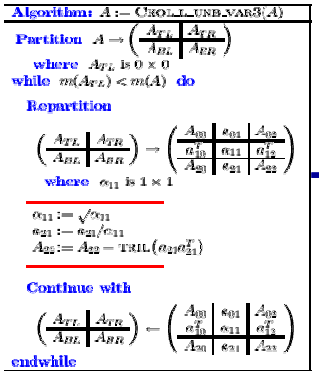
Exploiting task-level parallelism



- SuperMatrix: automatic identification of tasks/dependencies



Exploiting task-level parallelism



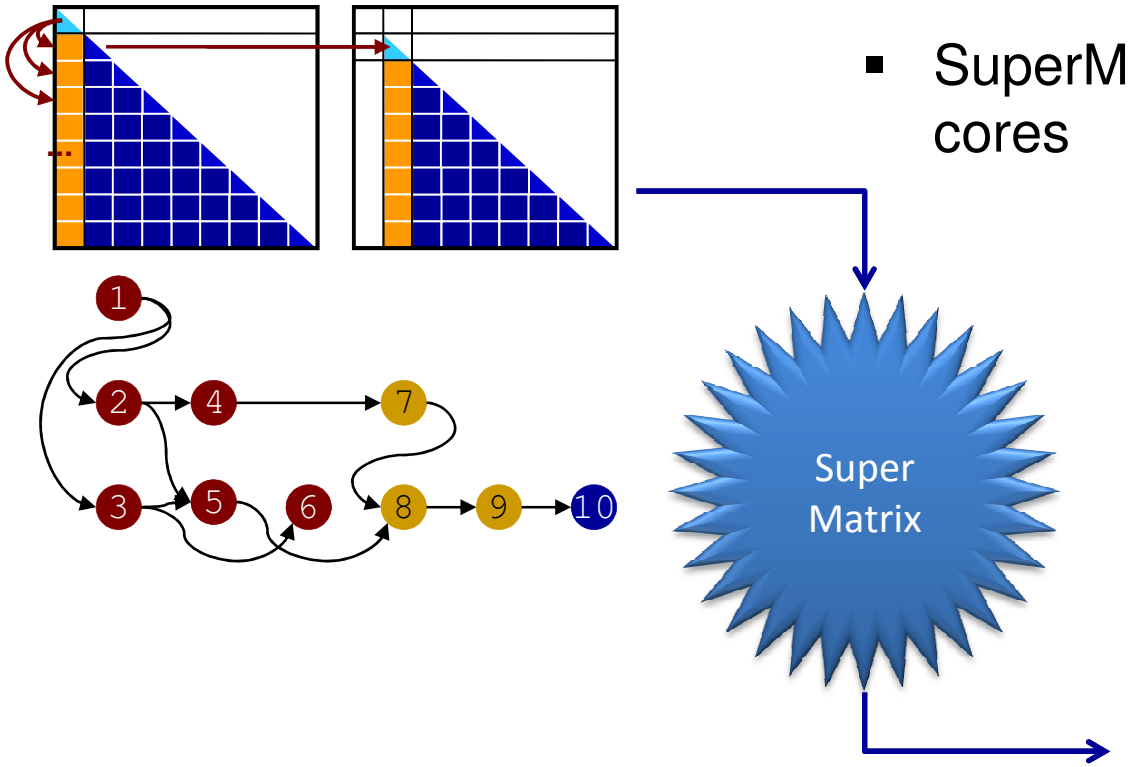

```
/*-----------------------------------*/
FLA_Chol( FLA_LOWER_TRIANGULAR, A11 );
FLA_Trsm( FLA_RIGHT, FLA_LOWER_TRIANGULAR,
          FLA_TRANSPOSE, FLA_NONUNIT_DIAG,
          FLA_ONE, A11, A21 );
FLA_Syrk( FLA_LOWER_TRIANGULAR,FLA_NO_TRANSPOSE,
          FLA_MINUS_ONE, A21, FLA_ONE, A22 );
/*-----------------------------------*/
```

- SuperMatrix: automatic identification of tasks/dependencies

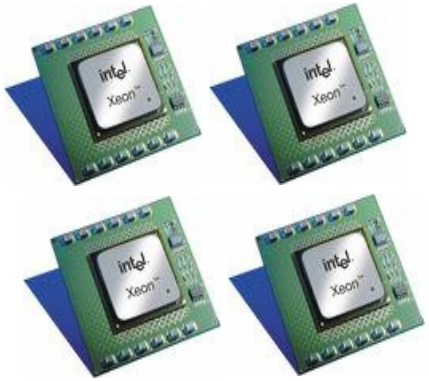
HOW?

- Input/output/input-output operands and order of operations in code determine dependencies
- Direction of operands is defined as part of BLAS specification

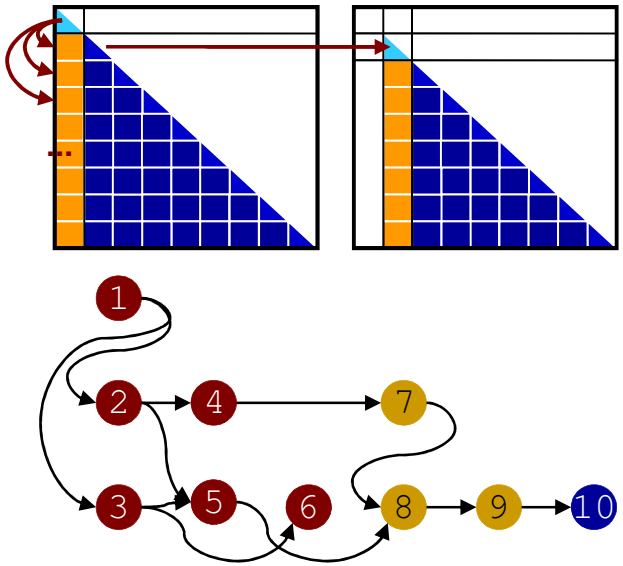
Exploiting task-level parallelism



- SuperMatrix: scheduling of tasks to cores



Exploiting task-level parallelism

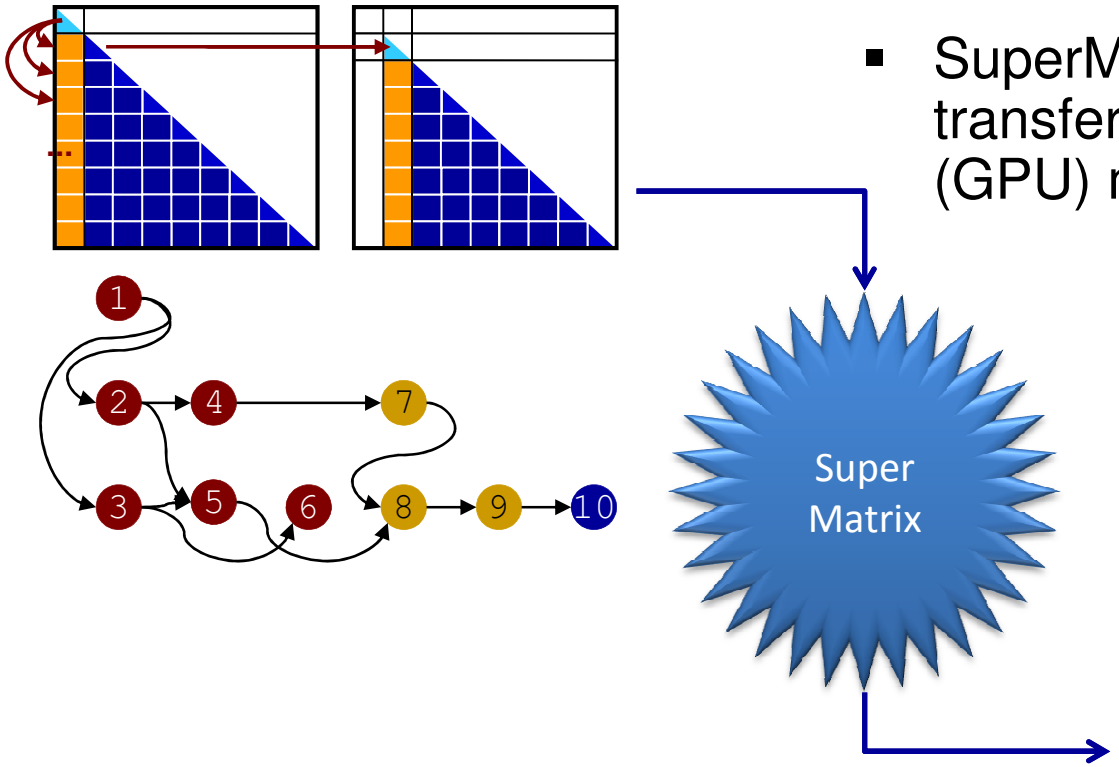


- SuperMatrix: scheduling of tasks to cores

HOW?

- List of ready tasks
- One thread per core
  1. Centralized list
  2. One list per-thread
  3. One list per-thread and work-stealing

Single GPU



- SuperMatrix: Dealing with data transfers between host (CPU)/device (GPU) memory spaces

Single GPU: a user's view

UNIVERSITAT JAUME·I
HPC&A

```
FLA_Obj A;

// Initialize conventional matrix: buffer, m, rs, cs
// Obtain storage blocksize, # of threads: b, n_threads

FLA_Init();

FLASH_Obj_create( FLA_DOUBLE, m, m, 1, &b, &A );
FLASH_Copy_buffer_to_hier( m, m, buffer, rs, cs,
                           0, 0, A );
FLASH_Queue_set_num_threads( n_threads );
FLASH_Queue_enable_gpu();
FLASH_Chol( FLA_LOWER_TRIANGULAR, A );

FLASH_Obj_free( &A );

FLA_Finalize();
```

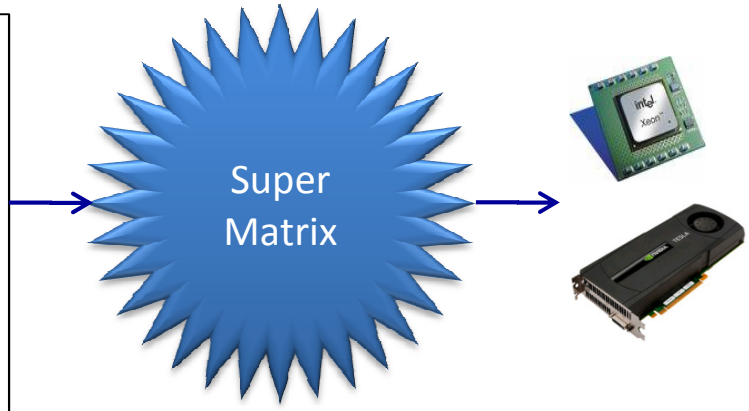
Single GPU: under the cover

```
FLA_Part_2x2(…);

while ( FLA_Obj_length(ATL) < FLA_Obj_length(A) ){

 FLA_Repart_2x2_to_3x3(…);

  /*--------------------------------------*/
  FLASH_Chol( FLA_LOWER_TRIANGULAR, A11 );
  FLASH_Trsm( FLA_RIGHT, FLA_LOWER_TRIANGULAR,
              FLA_TRANSPOSE, FLA_NONUNIT_DIAG,
              FLA_ONE, A11, A21 );
  FLASH_Syrk( FLA_LOWER_TRIANGULAR,FLA_NO_TRANSPOSE,
              FLA_MINUS_ONE, A21, FLA_ONE, A22 );
  /*--------------------------------------*/

  FLA_Cont_with_3x3_to_2x2(…);

}
```

Indexing operations (with addresses in device memory)

Single GPU: under the cover

```
FLA_Part_2x2(…);

while ( FLA_Obj_length(ATL) < FLA_Obj_length(A) ){

 FLA_Repart_2x2_to_3x3(…);

  /*-----------------------------------------*/
  FLASH_Chol( FLA_LOWER_TRIANGULAR, A11 );
  FLASH_Trsm( FLA_RIGHT, FLA_LOWER_TRIANGULAR,
              FLA_TRANSPOSE, FLA_NONUNIT_DIAG,
              FLA_ONE, A11, A21 );
  FLASH_Syrk( FLA_LOWER_TRIANGULAR,FLA_NO_TRANSPOSE,
              FLA_MINUS_ONE, A21, FLA_ONE, A22 );
  /*-----------------------------------------*/

  FLA_Cont_with_3x3_to_2x2(…);

}
```

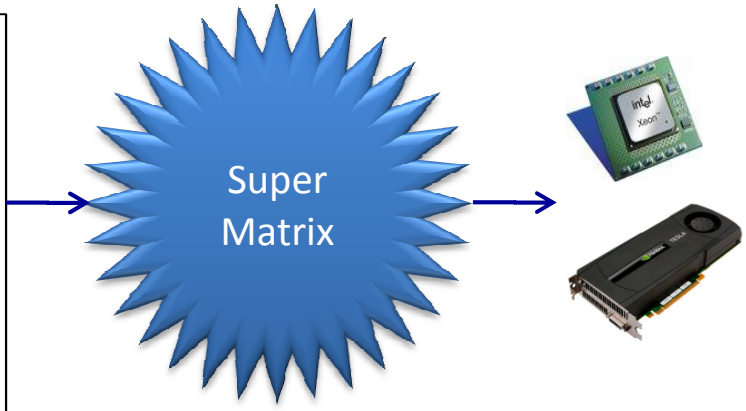


Real computation:

Runtime keeps track of data in host/device memory and performs the necessary transfers, reducing #copies

Single GPU: under the cover

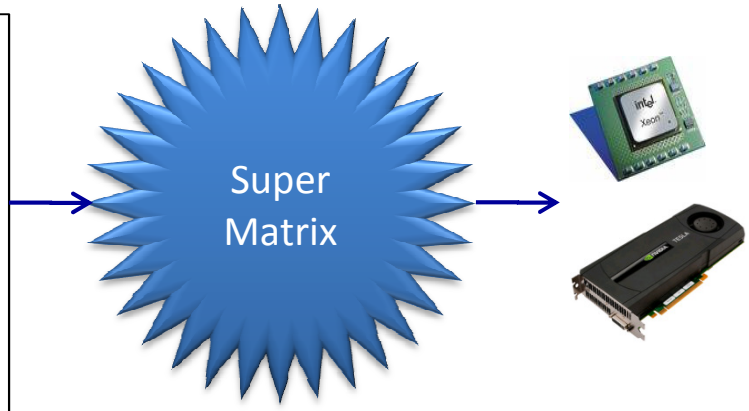
```
FLA_Part_2x2(…);

while ( FLA_Obj_length(ATL) < FLA_Obj_length(A) ){

 FLA_Repart_2x2_to_3x3(…);

  /*----------------------------------------*/
  FLASH_Chol( FLA_LOWER_TRIANGULAR, A11 );
  FLASH_Trsm( FLA_RIGHT, FLA_LOWER_TRIANGULAR,
              FLA_TRANSPOSE, FLA_NONUNIT_DIAG,
              FLA_ONE, A11, A21 );
  FLASH_Syrk( FLA_LOWER_TRIANGULAR,FLA_NO_TRANSPOSE,
              FLA_MINUS_ONE, A21, FLA_ONE, A22 );
  /*----------------------------------------*/

  FLA_Cont_with_3x3_to_2x2(…);

}
```



1. Copy matrix to GPU memory before algorithm commences

Single GPU: under the cover

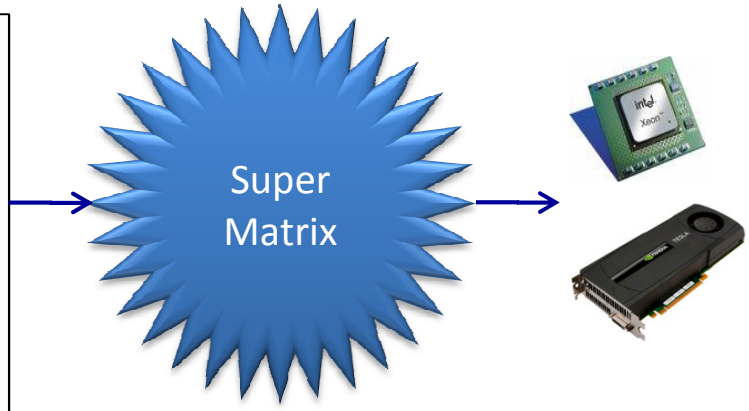
```
FLA_Part_2x2(…);

while ( FLA_Obj_length(ATL) < FLA_Obj_length(A) ){

 FLA_Repart_2x2_to_3x3(…);

  /*-----------------------------------------*/
  FLASH_Chol( FLA_LOWER_TRIANGULAR, A11 );
  FLASH_Trsm( FLA_RIGHT, FLA_LOWER_TRIANGULAR,
              FLA_TRANSPOSE, FLA_NONUNIT_DIAG,
              FLA_ONE, A11, A21 );
  FLASH_Syrk( FLA_LOWER_TRIANGULAR,FLA_NO_TRANSPOSE,
              FLA_MINUS_ONE, A21, FLA_ONE, A22 );
  /*-----------------------------------------*/

  FLA_Cont_with_3x3_to_2x2(…);

}
```



2. Copy block `A11` from device to host before its factorization

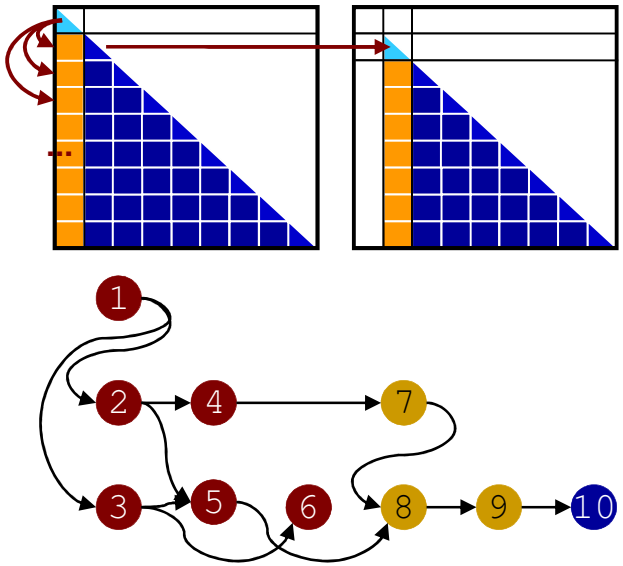
Single GPU: under the cover

```
FLA_Part_2x2(…);

while ( FLA_Obj_length(ATL) < FLA_Obj_length(A) ){

 FLA_Repart_2x2_to_3x3(…);

  /*----------------------------------------*/
  FLASH_Chol( FLA_LOWER_TRIANGULAR, A11 );
  FLASH_Trsm( FLA_RIGHT, FLA_LOWER_TRIANGULAR,
              FLA_TRANSPOSE, FLA_NONUNIT_DIAG,
              FLA_ONE, A11, A21 );
  FLASH_Syrk( FLA_LOWER_TRIANGULAR,FLA_NO_TRANSPOSE,
              FLA_MINUS_ONE, A21, FLA_ONE, A22 );
  /*----------------------------------------*/

  FLA_Cont_with_3x3_to_2x2(…);

}
```

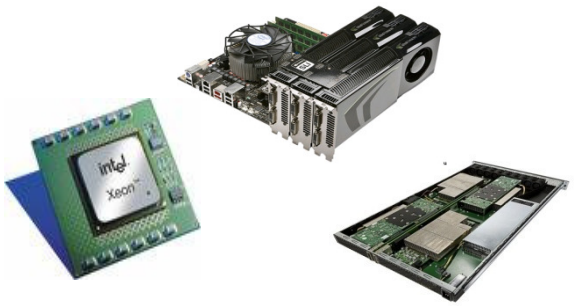


3. Copy block `A11` from host to device before using it in subsequent computations

Multi-GPU

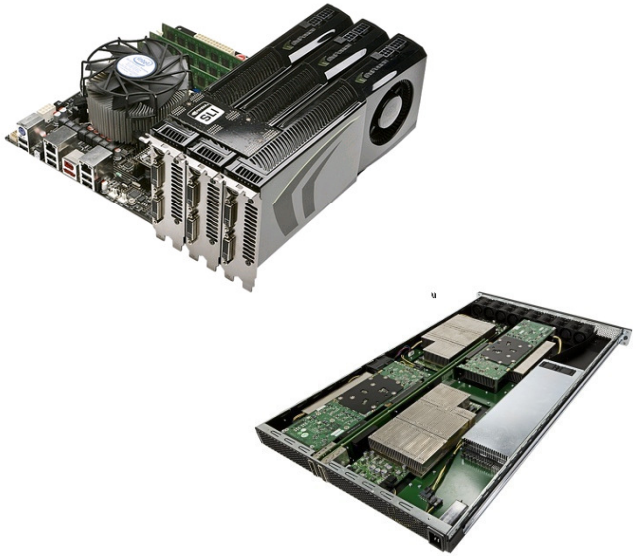


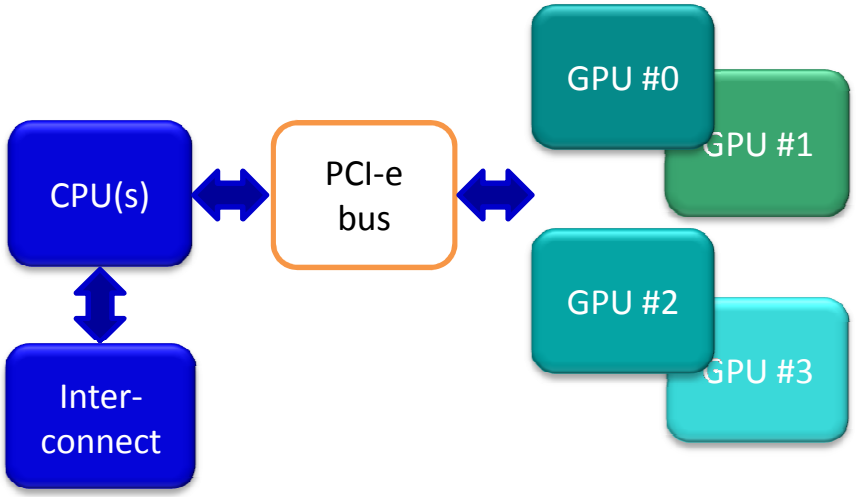
- SuperMatrix: Dealing with data transfers between host (CPU)/device (GPU) memory spaces



Multi-GPU

- How do we program these?





Multi-GPU: a user's view

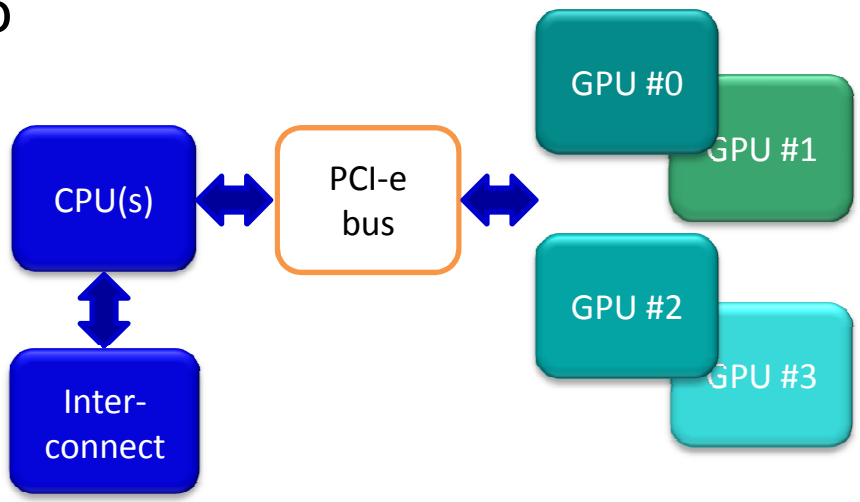
UNIVERSITAT JAUME·I HPC&A

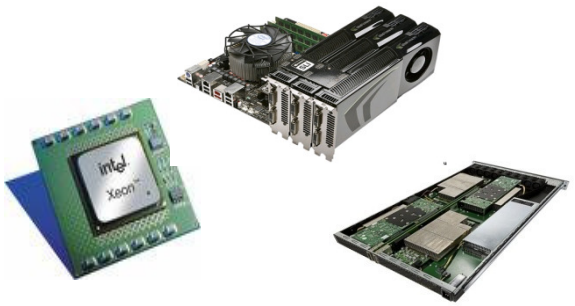
```
FLA_Obj A;

// Initialize conventional matrix: buffer, m, rs, cs
// Obtain storage blocksize, # of threads: b, n_threads

FLA_Init();

FLASH_Obj_create( FLA_DOUBLE, m, m, 1, &b, &A );
FLASH_Copy_buffer_to_hier( m, m, buffer, rs, cs,
                           0, 0, A );
FLASH_Queue_set_num_threads( n_threads );
FLASH_Queue_enable_gpu();
FLASH_Chol( FLA_LOWER_TRIANGULAR, A );

FLASH_Obj_free( &A );

FLA_Finalize();
```

Multi-GPU: under the cover

- Naïve approach:
- Before execution, transfer data to device
- Upon completion, retrieve results back to host

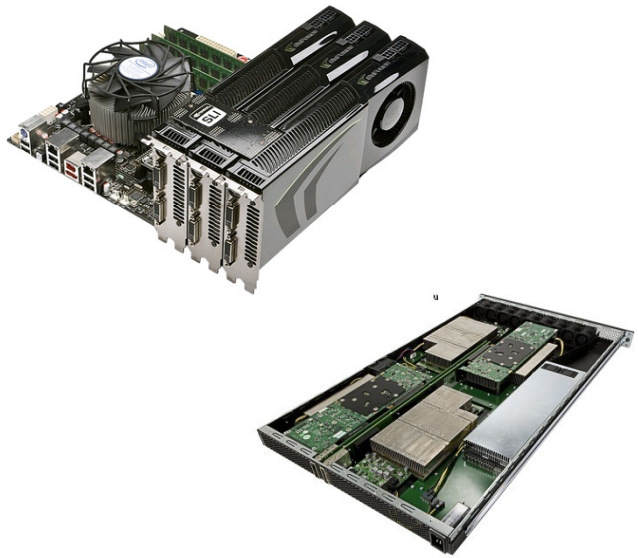
→ poor data locality

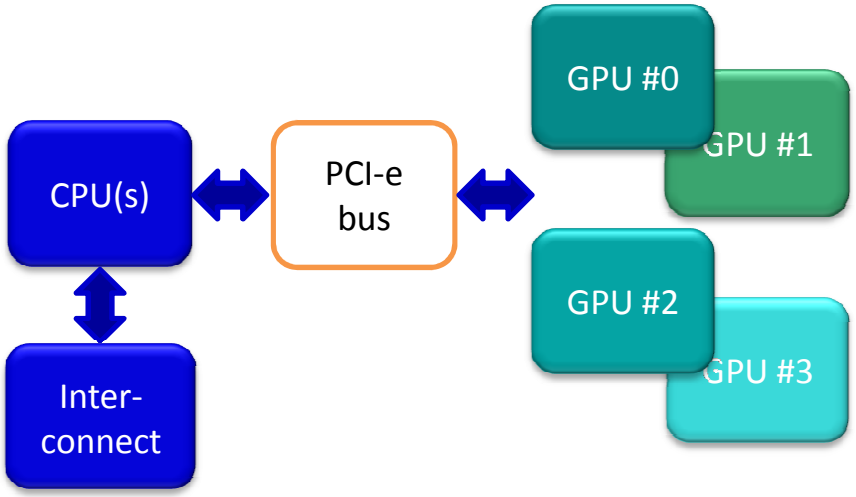




Multi-GPU: under the cover

- How do we program these?



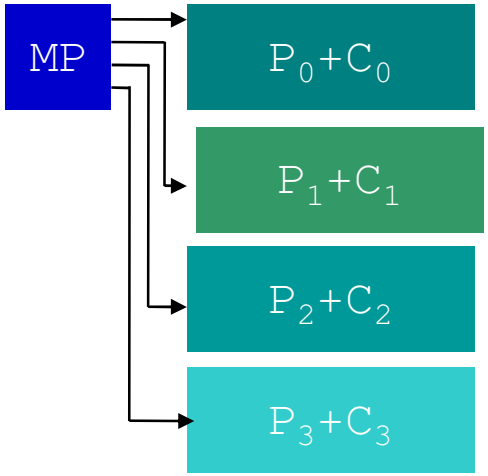


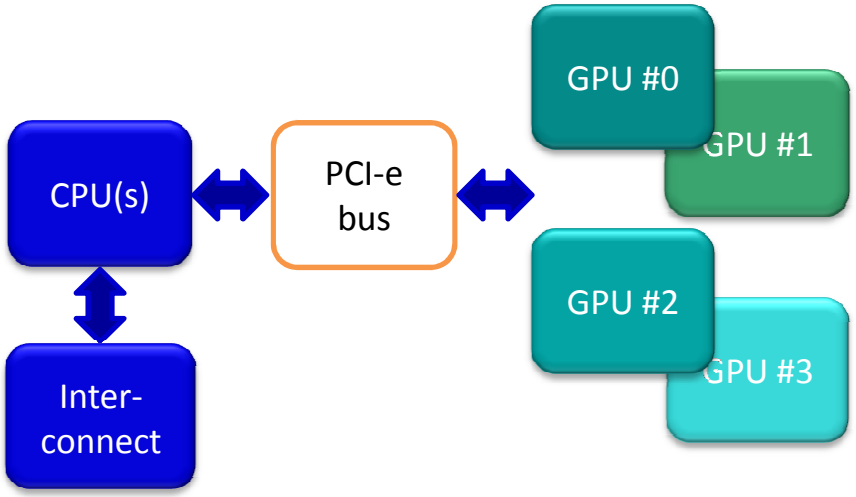
View as a…

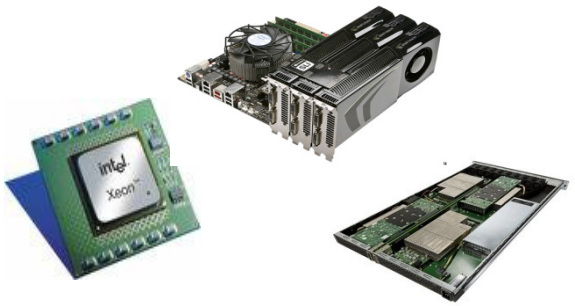
- Shared-memory multiprocessor + DSM

Multi-GPU: under the cover

- View system as a shared-memory multiprocessors (multi-core processor with hw. coherence)







Multi-GPU: under the cover

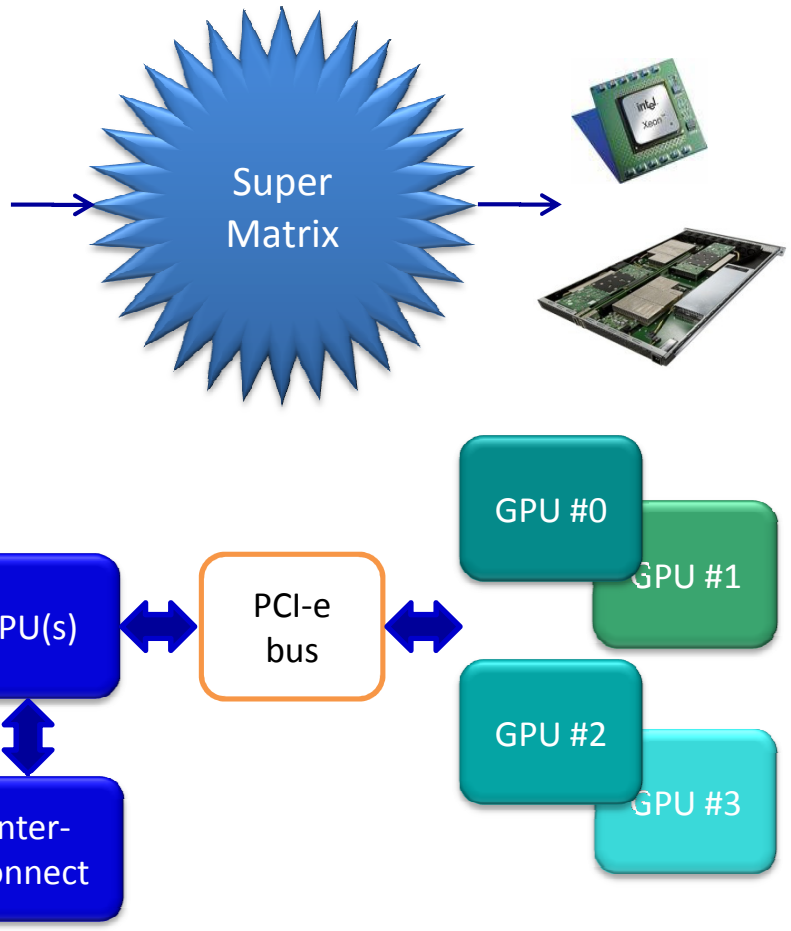
UNIVERSITAT JAUME·I HPC&A

- Software Distributed-Shared Memory (DSM)
  - Software: flexibility vs. efficiency
  - Underlying distributed memory hidden from the users
  - Reduce memory transfers using write-back, write-invalidate,…
  - Well-known approach, not too efficient as a middleware for general apps.

  - Regularity of dense linear algebra operations makes a difference!

Multi-GPU: under the cover

- Reduce #data transfers:
- Run-time handles device memory as a software cache:
  - Operate at block level
  - Software → flexibility
  - *Write-back*
  - *Write-invalidate*

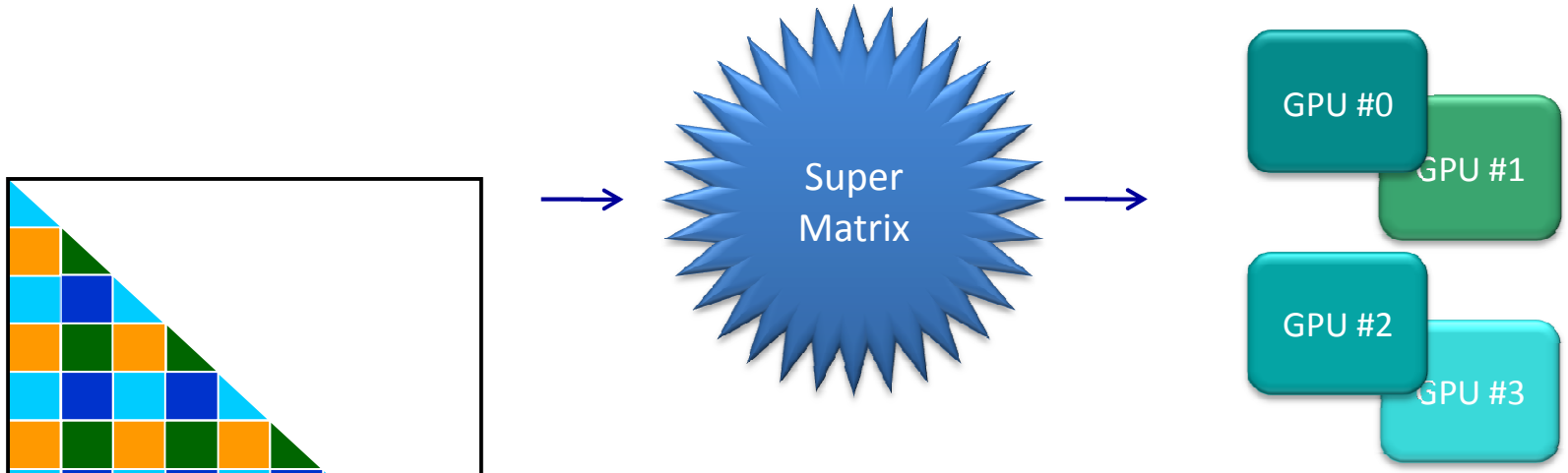


Multi-GPU: under the cover

```
FLA_Part_2x2(…);

while ( FLA_Obj_length(ATL) < FLA_Obj_length(A) ){

 FLA_Repart_2x2_to_3x3(…);

  /*----------------------------------------*/
  FLASH_Chol( FLA_LOWER_TRIANGULAR, A11 );
  FLASH_Trsm( FLA_RIGHT, FLA_LOWER_TRIANGULAR,
              FLA_TRANSPOSE, FLA_NONUNIT_DIAG,
              FLA_ONE, A11, A21 );
  FLASH_Syrk( FLA_LOWER_TRIANGULAR,FLA_NO_TRANSPOSE,
              FLA_MINUS_ONE, A21, FLA_ONE, A22 );
  /*----------------------------------------*/

  FLA_Cont_with_3x3_to_2x2(…);

}
```

1. Distribute matrix among GPU memories (2D workload distribution) before algorithm commences

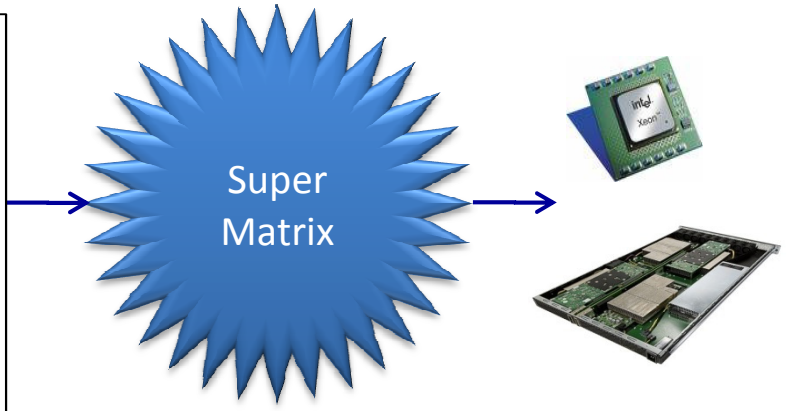
Multi-GPU: under the cover



1. Distribute matrix among GPU memories (2D workload distribution): owner-computes rule

Multi-GPU: under the cover

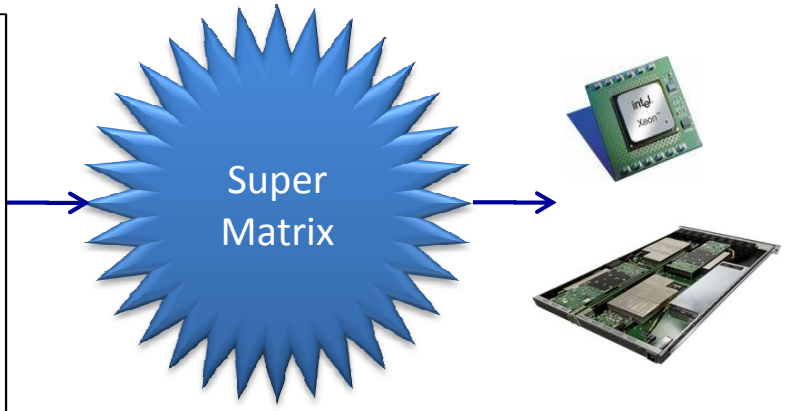
```
FLA_Part_2x2(…);

while ( FLA_Obj_length(ATL) < FLA_Obj_length(A) ){

 FLA_Repart_2x2_to_3x3(…);

  /*-----------------------------------------*/
  FLASH_Chol( FLA_LOWER_TRIANGULAR, A11 );
  FLASH_Trsm( FLA_RIGHT, FLA_LOWER_TRIANGULAR,
              FLA_TRANSPOSE, FLA_NONUNIT_DIAG,
              FLA_ONE, A11, A21 );
  FLASH_Syrk( FLA_LOWER_TRIANGULAR,FLA_NO_TRANSPOSE,
              FLA_MINUS_ONE, A21, FLA_ONE, A22 );
  /*-----------------------------------------*/

  FLA_Cont_with_3x3_to_2x2(…);

}
```



2. Copy block `A11` from corresponding device to host before its factorization

Multi-GPU: under the cover

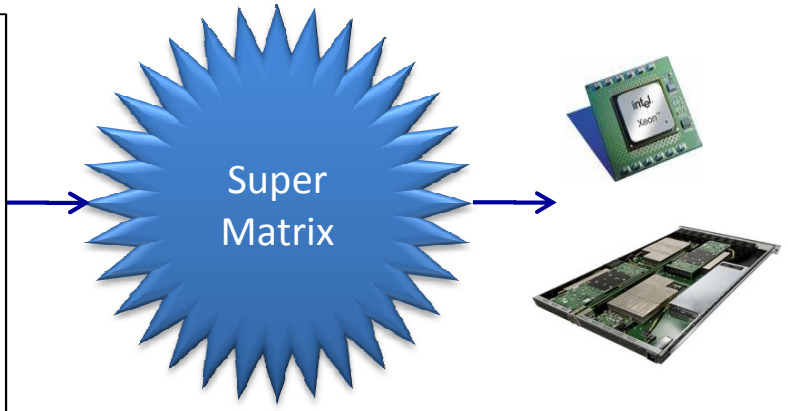
```
FLA_Part_2x2(…);

while ( FLA_Obj_length(ATL) < FLA_Obj_length(A) ){

 FLA_Repart_2x2_to_3x3(…);

  /*----------------------------------------*/
  FLASH_Chol( FLA_LOWER_TRIANGULAR, A11 );
  FLASH_Trsm( FLA_RIGHT, FLA_LOWER_TRIANGULAR,
              FLA_TRANSPOSE, FLA_NONUNIT_DIAG,
              FLA_ONE, A11, A21 );
  FLASH_Syrk( FLA_LOWER_TRIANGULAR,FLA_NO_TRANSPOSE,
              FLA_MINUS_ONE, A21, FLA_ONE, A22 );
  /*----------------------------------------*/

  FLA_Cont_with_3x3_to_2x2(…);

}
```



3. Broadcast block `A11` from host to appropriate devices before using it in subsequent computations (write-update)

Multi-GPU: under the cover

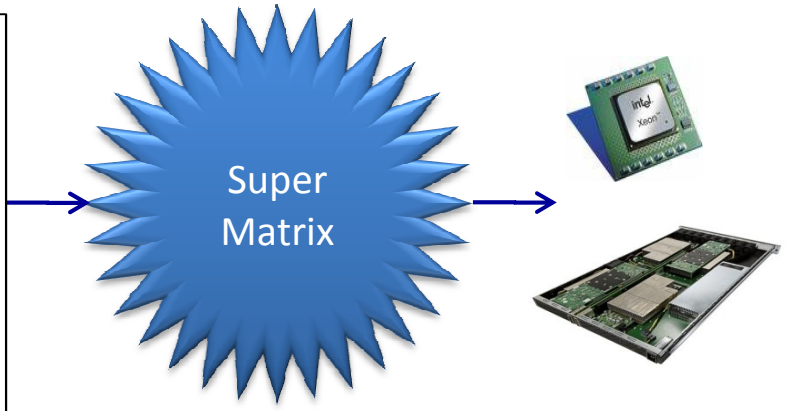
```
FLA_Part_2x2(…);

while ( FLA_Obj_length(ATL) < FLA_Obj_length(A) ){

 FLA_Repart_2x2_to_3x3(…);

  /*----------------------------------------*/
  FLASH_Chol( FLA_LOWER_TRIANGULAR, A11 );
  FLASH_Trsm( FLA_RIGHT, FLA_LOWER_TRIANGULAR,
              FLA_TRANSPOSE, FLA_NONUNIT_DIAG,
              FLA_ONE, A11, A21 );
  FLASH_Syrk( FLA_LOWER_TRIANGULAR,FLA_NO_TRANSPOSE,
              FLA_MINUS_ONE, A21, FLA_ONE, A22 );
  /*----------------------------------------*/

  FLA_Cont_with_3x3_to_2x2(…);

}
```



4. Keep `A11` in receiving device(s) in case needed in subsequent computations (cache)

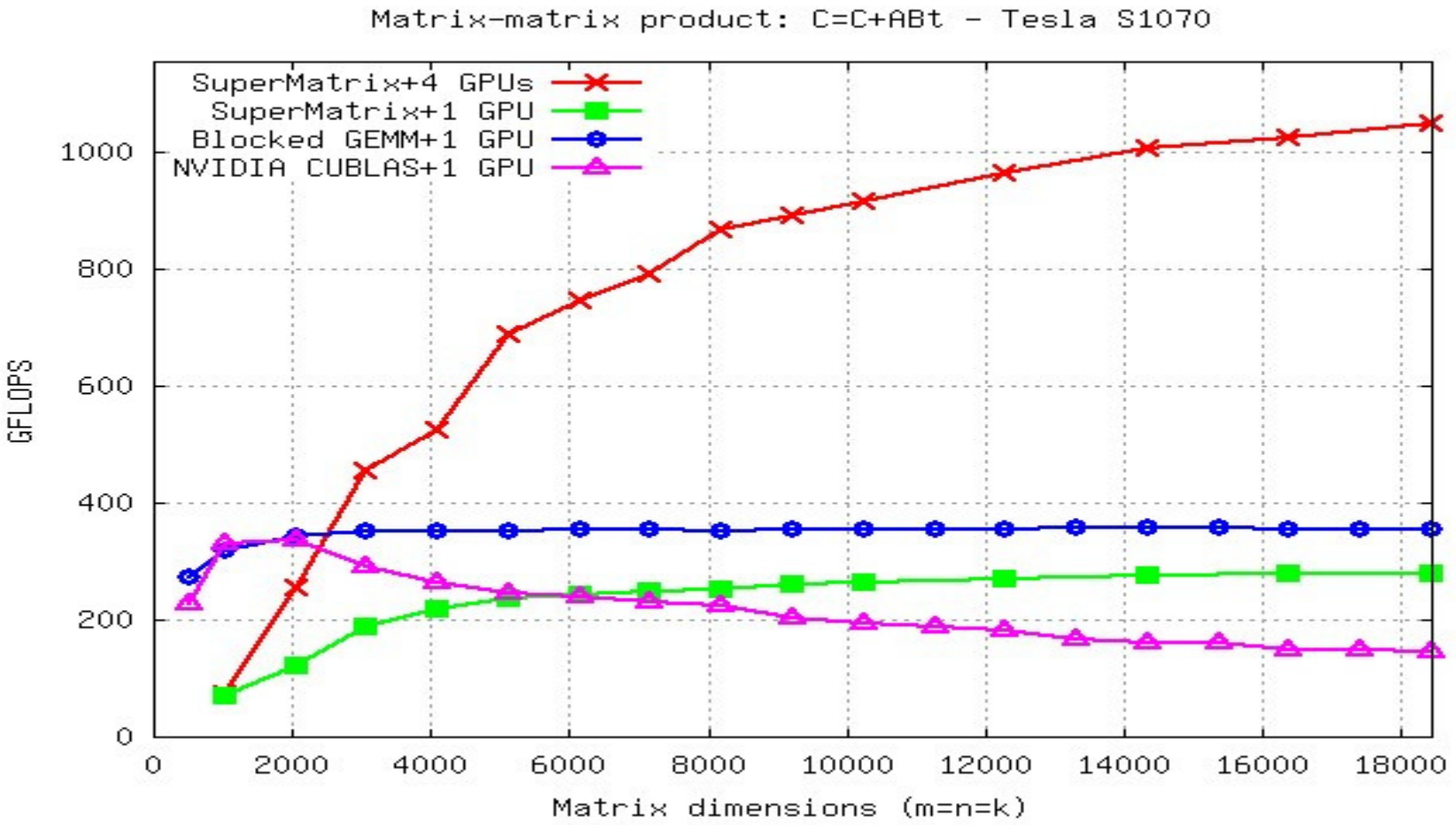
Multi-GPU: under the cover

```
FLA_Part_2x2(…);

while ( FLA_Obj_length(ATL) < FLA_Obj_length(A) ){

 FLA_Repart_2x2_to_3x3(…);

  /*----------------------------------------*/
  FLASH_Chol( FLA_LOWER_TRIANGULAR, A11 );
  FLASH_Trsm( FLA_RIGHT, FLA_LOWER_TRIANGULAR,
              FLA_TRANSPOSE, FLA_NONUNIT_DIAG,
              FLA_ONE, A11, A21 );
  FLASH_Syrk( FLA_LOWER_TRIANGULAR,FLA_NO_TRANSPOSE,
              FLA_MINUS_ONE, A21, FLA_ONE, A22 );
  /*----------------------------------------*/

  FLA_Cont_with_3x3_to_2x2(…);

}
```

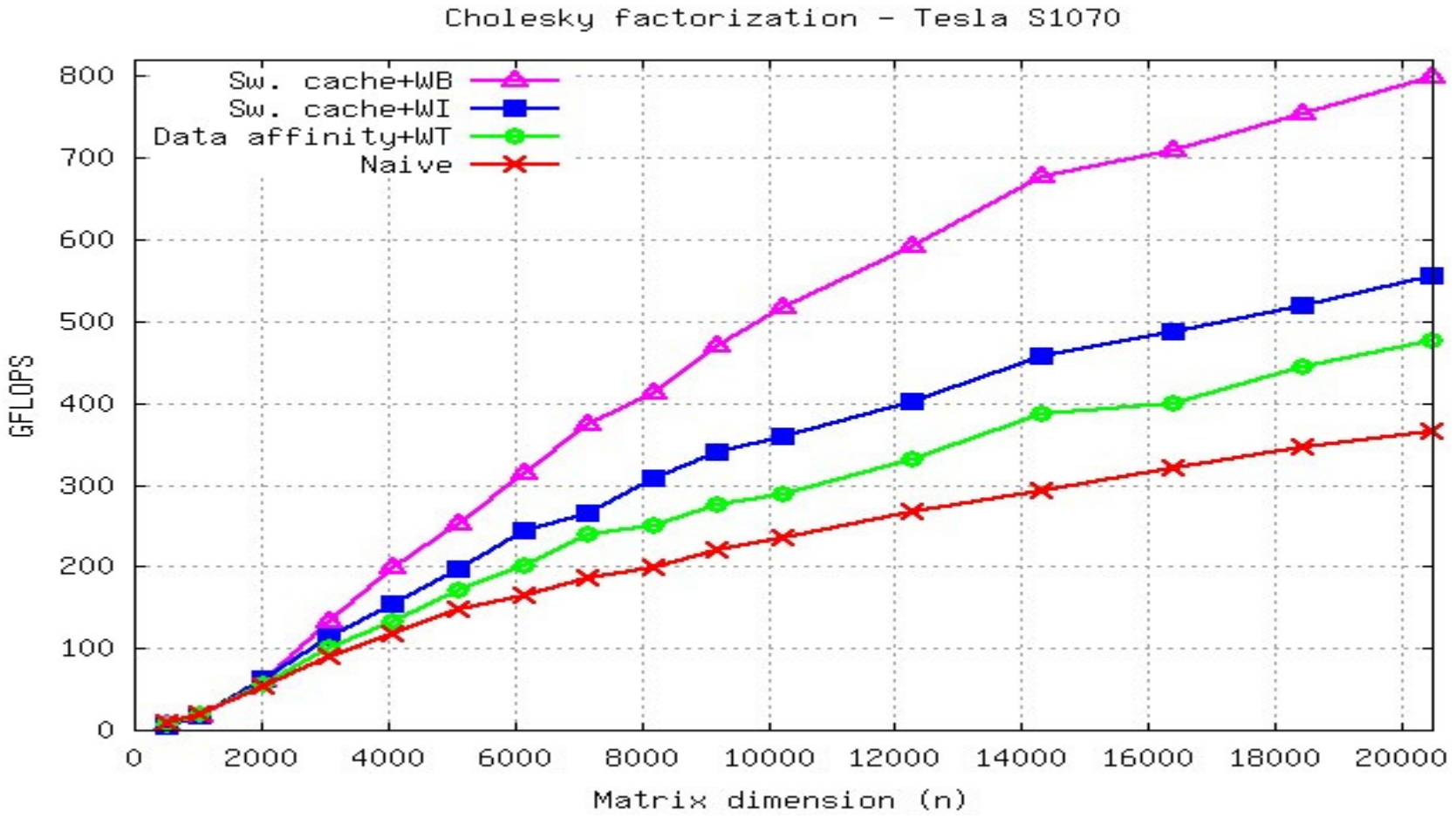


5. Keep updated `A21` in device till replaced (write-back)

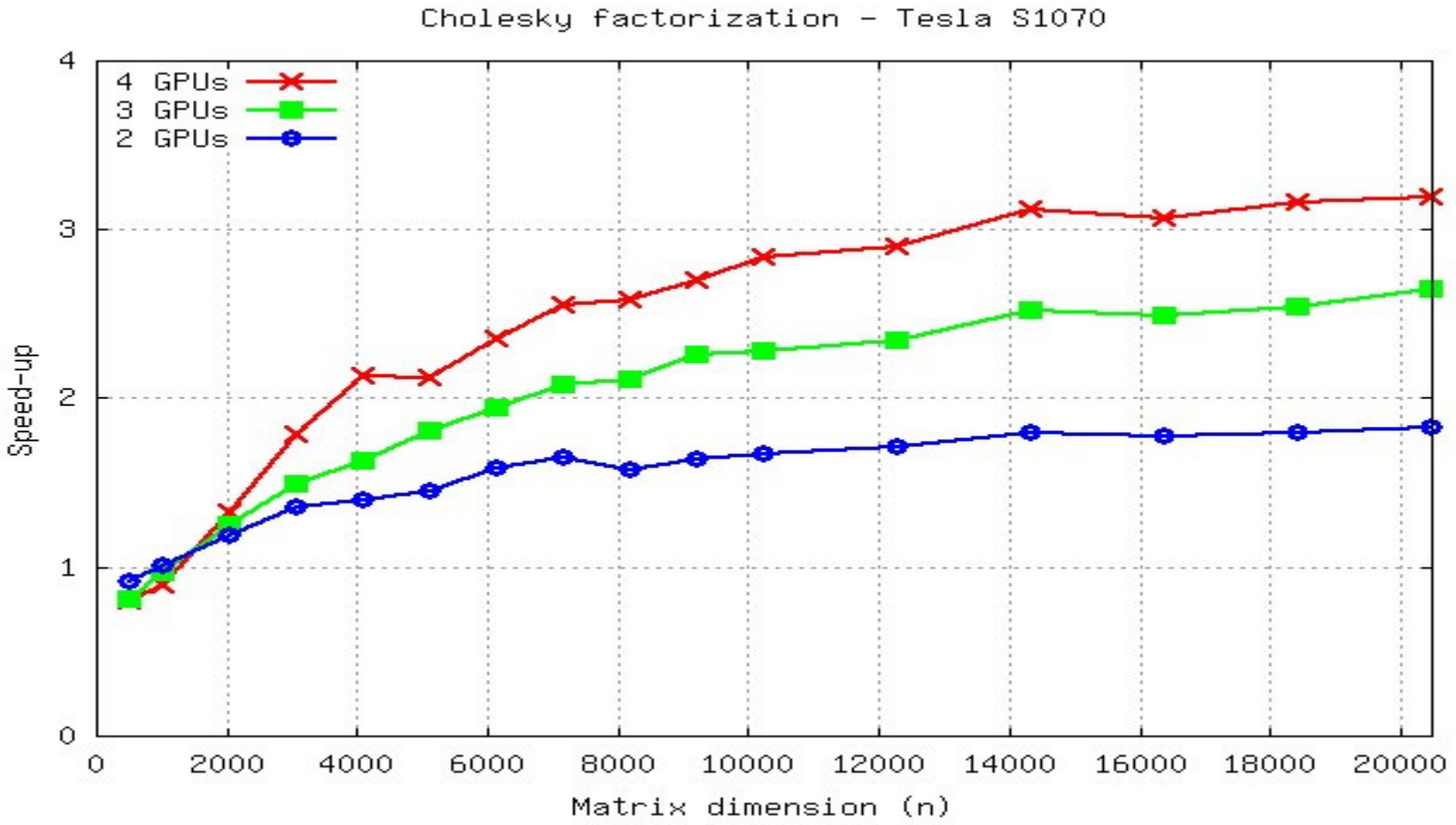
Performance



Performance



Performance



Index

UNIVERSITAT JAUME·I
HPC&A

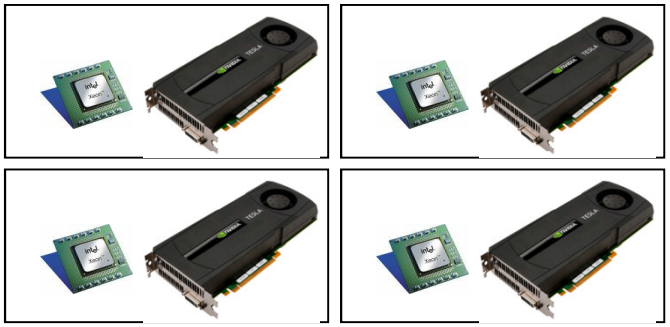
- The libflame library
  1. A user's view
  2. Creating your own algorithm
  3. FLAME runtime
  4. Clusters of GPUs
     1. DLA for clusters
     2. Host-centric view
     3. Device-centric view
- The StarSs framework

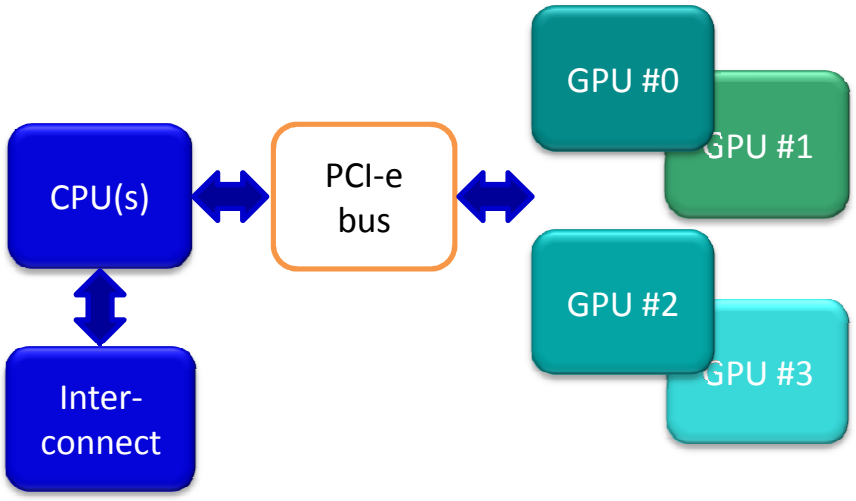
libflame-like libraries

- PLAPACK (UT@Austin)
  - Use of objects (`PLA_Obj`), vectors, matrices, projected vectors, etc., with layout embedded
  - PMB distribution
  - Layered and modular design: all communication is done via copies (`PLA_Copy`) and reductions (`PLA_Reduce`) from one object type to another
- Elemental (Jack Poulson)
  - Based on PLAPACK, but C++
  - Element-wise cyclic data layout

Data in host memory

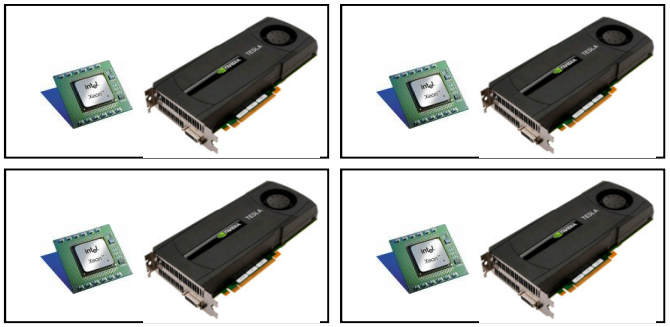
- Before executing a kernel, copy input data to GPU memory
- After execution, retrieve results back to node main memory
- Easy to program (wrappers to kernels)
- Copies linked to kernel execution: $O(n^3)$ transfers between CPU and GPU

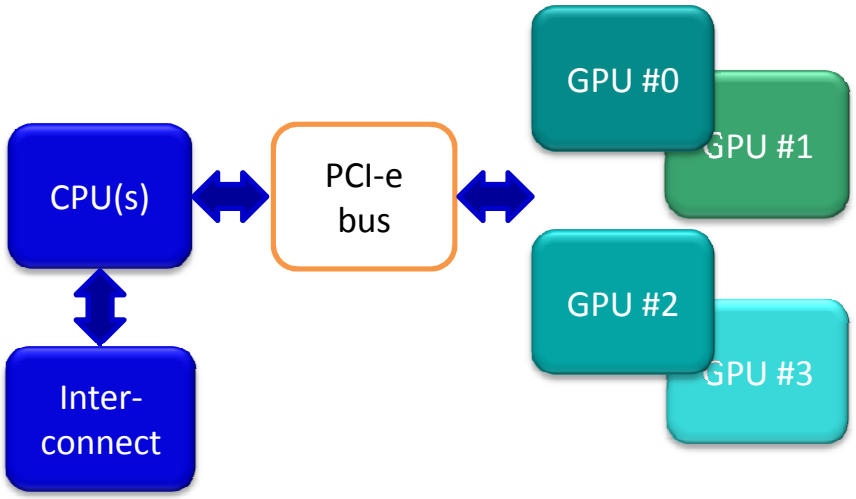




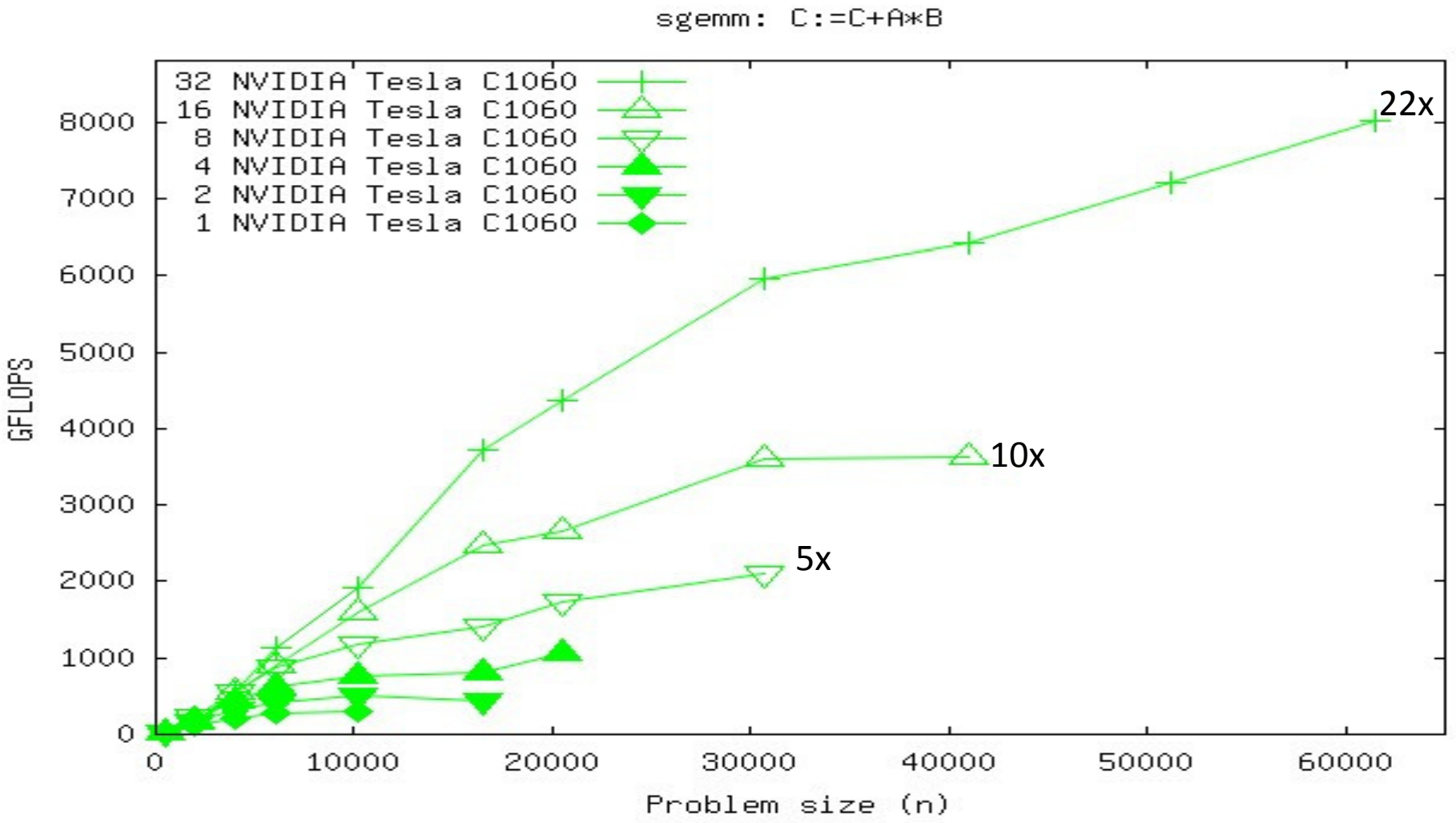
Data in GPU memory

- Before sending a piece of data, retrieve it back to node main memory (compact on the fly)
- After reception, copy contents to GPU memory
- Easy to program (wrappers to MPI calls)
- Copies linked to communication, not kernel execution: $O(n^2)$ transfers between CPU and GPU

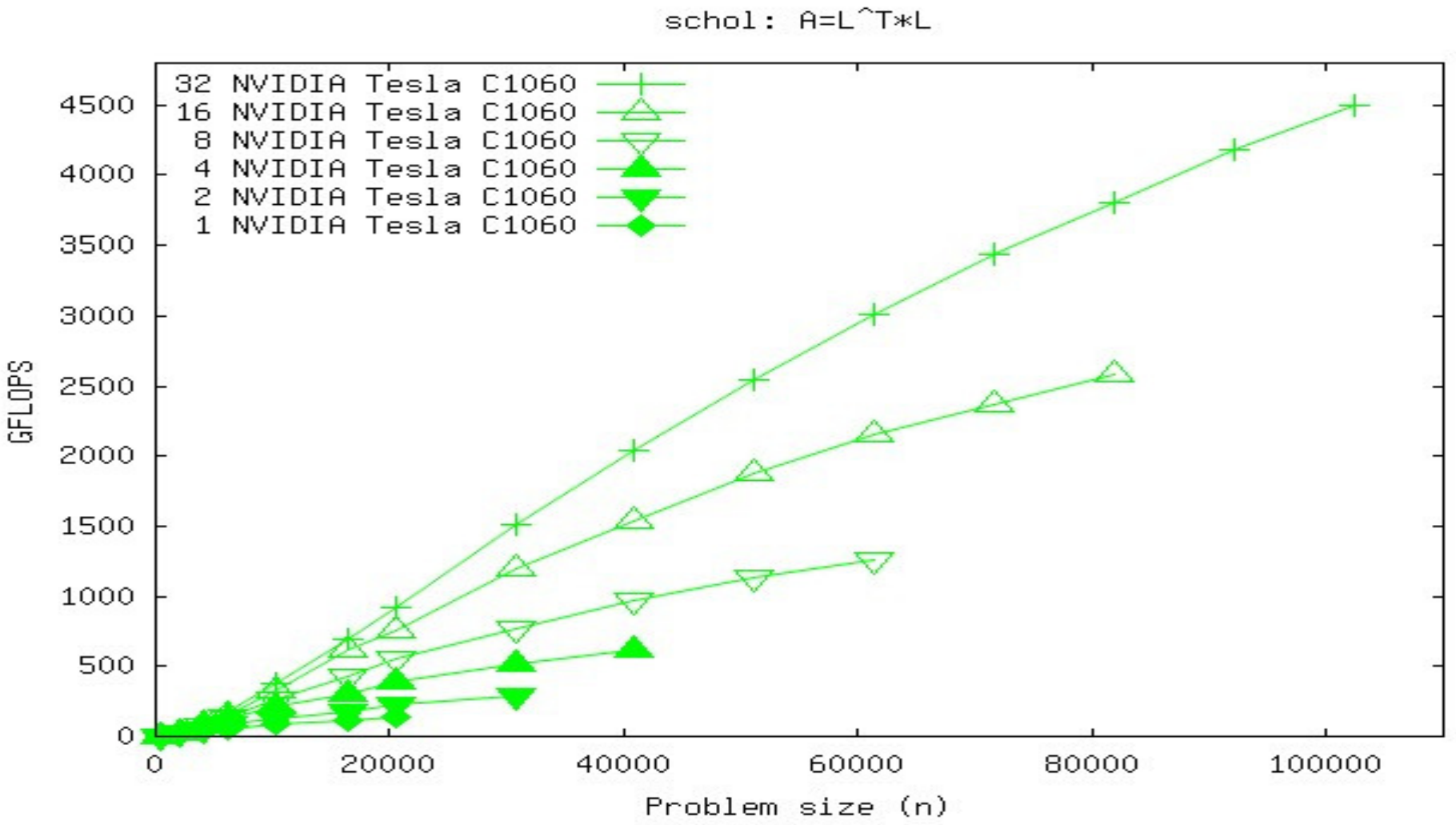




Performance



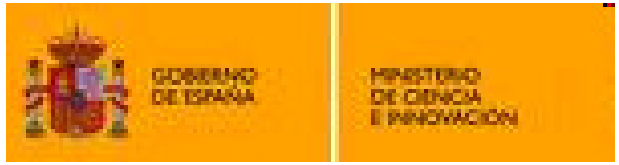
Performance



Acknowledgements

UNIVERSITAT JAUME·I
HPC&A

- Funding sources



Further information

UNIVERSITAT JAUME·I
HPC&A

- Contact:
  - field@cs.utexas.edu
- FLAME project website:
  - www.cs.utexas.edu/users/flame/
- *libflame: The Complete Reference*
  - www.cs.utexas.edu/users/field/docs/
    - Updated nightly
  - www.lulu.com/content/5915632
    - Updated occasionally

Index

- The libflame library

- The StarSs framework

GPU support

The TEXT project

UNIVERSITAT JAUME·I HPC&A

- Towards Exaflop applicaTions

- Demonstrate that **Hybrid MPI/SMPSs** addresses the Exascale challenges in a productive and efficient way.
  - Deploy at supercomputing centers: Julich, EPCC, HLRS, BSC
  - Port Applications (HLA, SPECFEM3D, PEPC, PSC, BEST, CPMD, LS1 MarDyn) and develop algorithms.
  - Develop additional environment capabilities
    - tools (debug, performance)
    - improvements in runtime systems (load balance and GPUSs)
  - Support other users
    - Identify users of TEXT applications
    - Identify and support interested application developers
  - Contribute to Standards (OpenMP ARB, PERI-XML)

Index

UNIVERSITAT JAUME·I
HPC&A

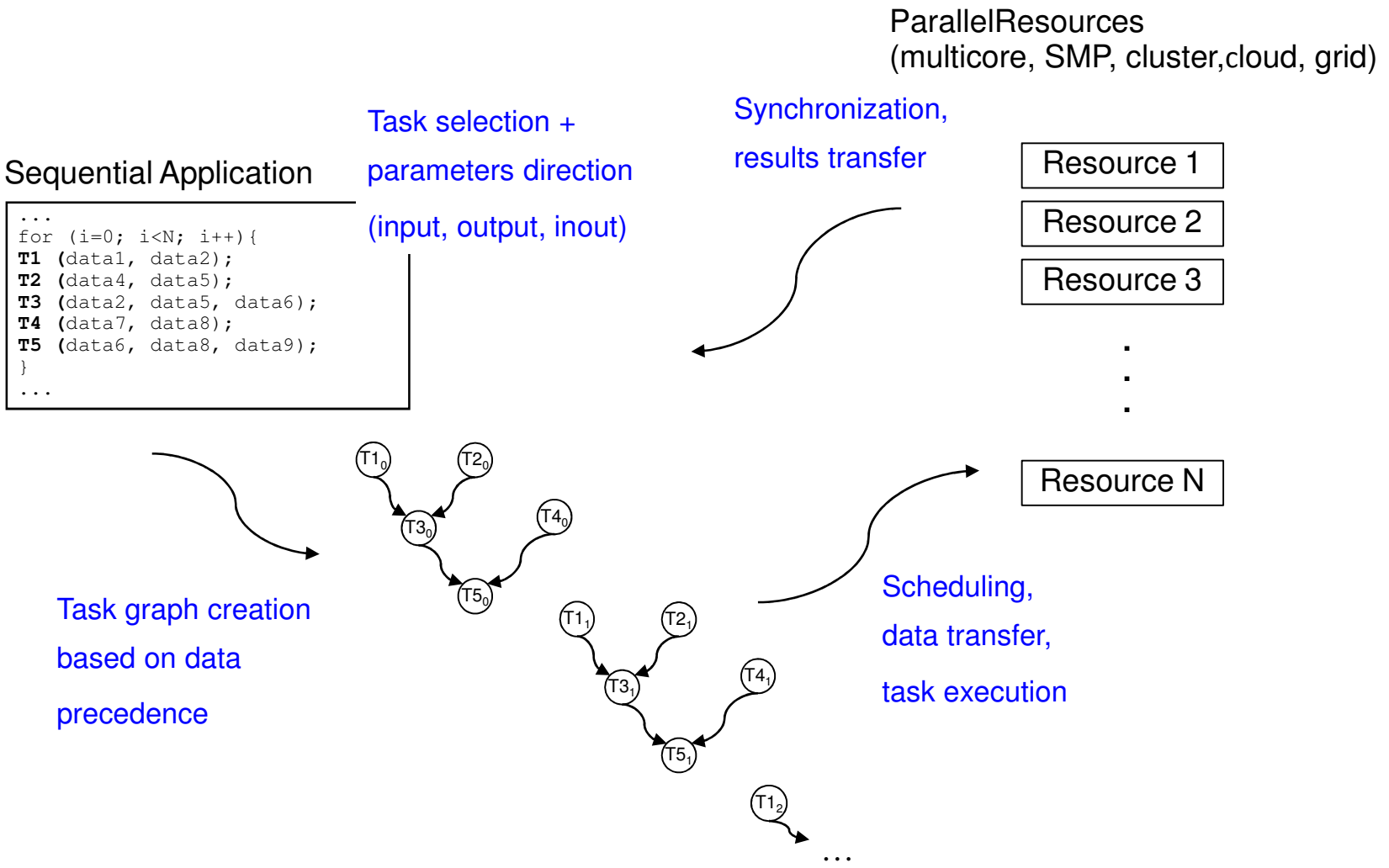
- The libflame library

- The StarSs framework

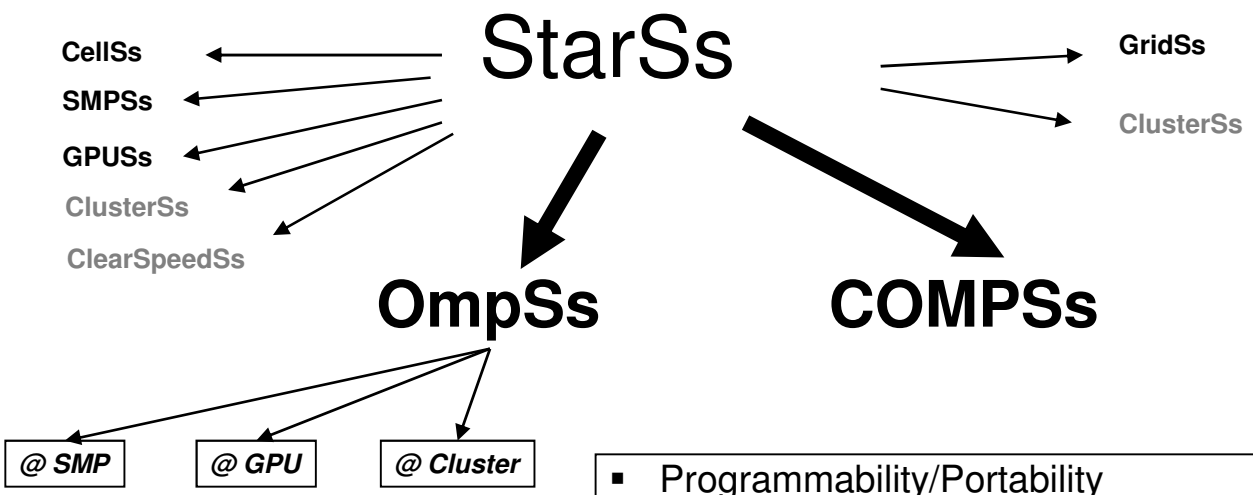
1. StarSs overview
2. OmpSs

Slides from Rosa M. Badia
Barcelona Supercomputing Center
Thanks!

Programming model



Programming model



- StarSs
  - A "node" level programming model
  - Sequential C/Fortran/Java + annotations
  - Task based. Asynchrony, data-flow.
  - "Simple" linear address space
  - Directionality annotations on tasks arguments
  - Nicely integrates in hybrid MPI/StarSs
  - Natural support for heterogeneity

- Programmability/Portability
  - Incremental parallelization/restructure
  - Separate algorithm from resources
  - Disciplined programming
  - **"Same" source code runs on "any" machine**
    - Optimized task implementations will result in better performance.
- Performance
  - Intelligent Runtime
    - Automatically extracts and exploits parallelism
      - Dataflow, workflow
    - Matches computations to specific resources on each type of target platform
  - Asynchronous (data-flow) execution and locality awareness

A sequential program…

UNIVERSITAT JAUME·I
HPC&A

```
void vadd3 (float A[BS], float B[BS],
            float C[BS]);

void scale_add (float sum, float A[BS],
                float B[BS]);

void accum (float A[BS], float *sum);
```

```
for (i=0; i<N; i+=BS)              // C=A+B
   vadd3 ( &A[i], &B[i], &C[i]);
...
for (i=0; i<N; i+=BS)             // sum(C[i])
   accum (&C[i], &sum);
...
for (i=0; i<N; i+=BS)             // B=sum*A
   scale_add (sum, &E[i], &B[i]);
...
for (i=0; i<N; i+=BS)             // A=C+D
   vadd3 (&C[i], &D[i], &A[i]);
...
for (i=0; i<N; i+=BS)             // E=G+F
   vadd3 (&G[i], &F[i], &E[i]);
```

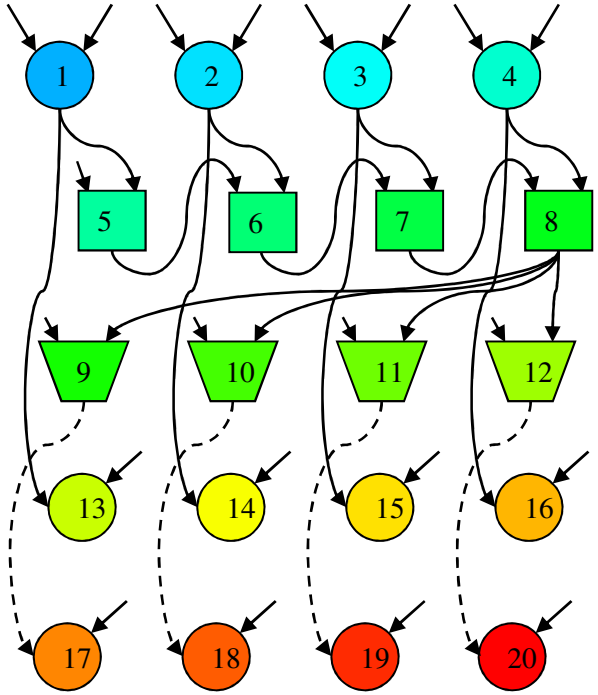
A sequential program… taskified…

UNIVERSITAT JAUME·I HPC&A

```
#pragma css task input(A, B) output(C)
void vadd3 (float A[BS], float B[BS],
            float C[BS]);
#pragma css task input(sum, A) inout(B)
void scale_add (float sum, float A[BS],
                float B[BS]);
#pragma css task input(A) inout(sum)
void accum (float A[BS], float *sum);
```

```
for (i=0; i<N; i+=BS)              // C=A+B
   vadd3 ( &A[i], &B[i], &C[i]);
...
for (i=0; i<N; i+=BS)              // sum(C[i])
   accum (&C[i], &sum);
...
for (i=0; i<N; i+=BS)              // B=sum*A
   scale_add (sum, &E[i], &B[i]);
...
for (i=0; i<N; i+=BS)              // A=C+D
   vadd3 (&C[i], &D[i], &A[i]);
...
for (i=0; i<N; i+=BS)              // E=G+F
   vadd3 (&G[i], &F[i], &E[i]);
```

Compute dependences @ task instantiation time



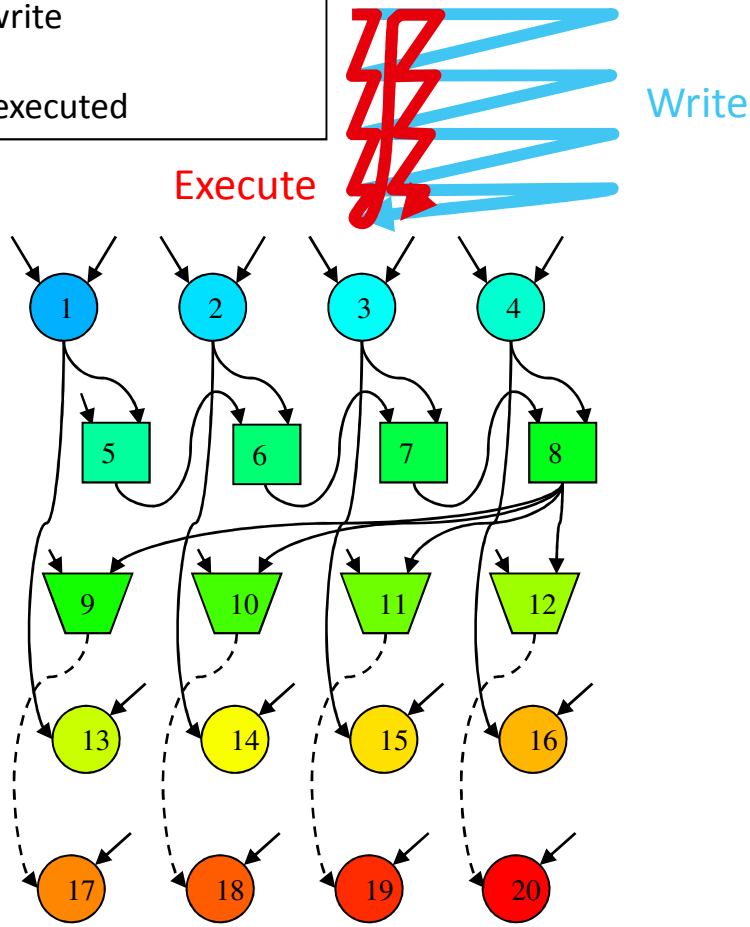
Color/number: order of task instantiation
Some antidependences covered by flow dependences not drawn

A sequential program… taskified… with data-flow execution

```
#pragma css task input(A, B) output(C)
void vadd3 (float A[BS], float B[BS],
            float C[BS]);
#pragma css task input(sum, A) inout(B)
void scale_add (float sum, float A[BS],
                float B[BS]);
#pragma css task input(A) inout(sum)
void accum (float A[BS], float *sum);
```

```
for (i=0; i<N; i+=BS)              // C=A+B
   vadd3 ( &A[i], &B[i], &C[i]);
...
for (i=0; i<N; i+=BS)              // sum(C[i])
   accum (&C[i], &sum);
...
for (i=0; i<N; i+=BS)              // B=sum*A
   scale_add (sum, &E[i], &B[i]);
...
for (i=0; i<N; i+=BS)              // A=C+D
   vadd3 (&C[i], &D[i], &A[i]);
...
for (i=0; i<N; i+=BS)              // E=G+F
   vadd3 (&G[i], &F[i], &E[i]);
```

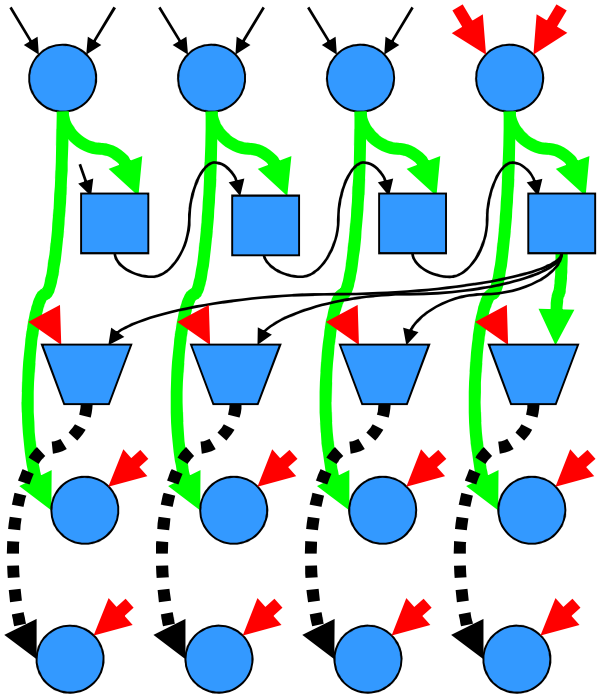
Decouple
how we write
from
how it is executed



Color/number: a possible order of task execution

The potential of data access information

- Flexibility to dynamically traverse dataflow graph "optimizing"
  - Concurrency. Critical path
  - Memory access: data transfers performed by run time
- Opportunities for
  - Prefetch
  - Reuse
  - Eliminate antidependences (rename)
  - Replication management
    - Coherency/consistency handled by the runtime



Index

- The libflame library

- The StarSs framework

1. StarSs overview
2. OmpSs

→

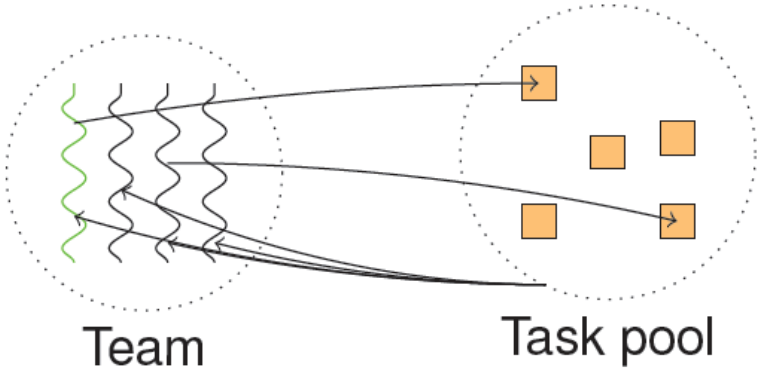
1. Overview & syntax
2. Compiler
3. Runtime
4. Examples

OmpSs = OpenMP + StarSs extensions

- OmpSs is based on OpenMP with some differences:
  - Different execution model
  - Extended memory model
  - Extensions for point-to-point inter-task synchronizations
    - data dependencies
  - Extensions for heterogeneity
  - Other minor extensions

Execution model

- Thread-pool model
  - OpenMP parallel “ignored”
- All threads created on startup
  - One of them starts executing main
- All get work from a task pool
  - And can generate new work



Memory model

- Two “modes“ are allowed:
  - pure SMP:
    - Single address space
    - OpenMP standard memory model is used
  - non-SMP (cluster, GPUs, ...):
    - Multiple address spaces exists
      - Same data may exists in multiple of these
    - Data consistency ensured by the implementation

Main element: Task

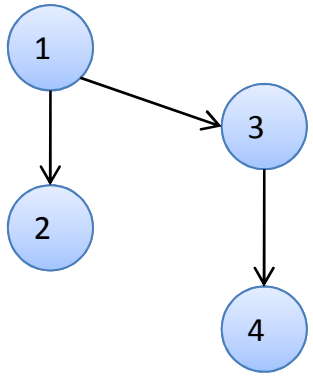
- Task: unit of computation
- Task definition
  - Pragmas in lined
  - Pragmas attached to function definition

```
#pragma omp task
void foo (int Y[size], int size) {
  int j;

  for (j=0; j<size; j++) Y[j]= j;
}

int main()
{
  int X[100]
  foo (X, 100);
}
```

Defining dependences

- Clauses that express data direction:
  - input
  - output
  - inout
- Dependences computed at runtime taking into account these clauses

```
#pragma omp task output( x )
x = 5;
#pragma omp task input( x )
printf("%d\n" , x ) ;
#pragma omp task inout( x )
x++;
#pragma omp task input( x )
printf ("%d\n" , x ) ;
```



Heterogeneity: the target directive

- Directive to specify device specific information:

  #pragma omp target [clauses]

  - Clauses:
    - device: which device (smp, gpu)
    - copy_in, copy_out, copy_inout: data to be moved in and out
    - implements: specifies alternate implementations

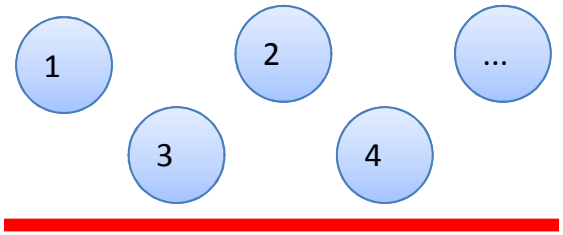
```
#pragma target device (smp)
#pragma omp task input (Y)
void foo (int Y[size], int size) {
  int j;

  for (j=0; j<size; j++) Y[j]= j;
}

int main()
{
  int X[100]
  foo (X, 100) ;
}
```

Synchronization

#pragma omp task wait

- Suspends the current task until all children tasks are completed
- Just direct children, not descendants

```
void traverse_list ( List l )
{
  Element e ;
  for ( e = l-> first; e ; e = e->next )
    #pragma omp task
    process ( e ) ;

  #pragma omp taskwait
}
```

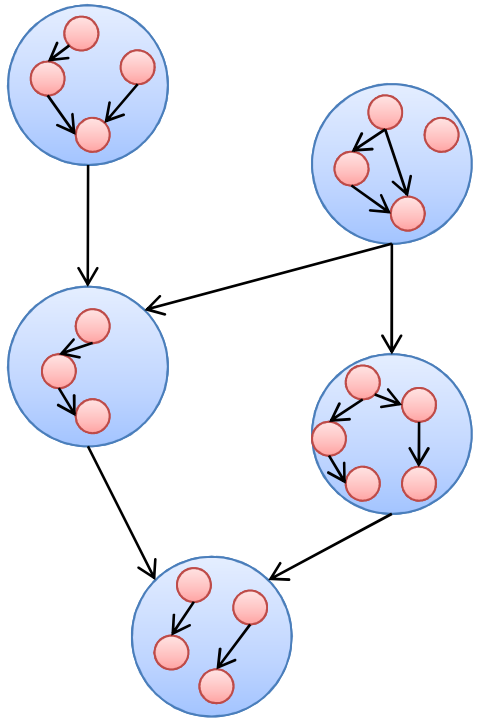


Hierarchical task graph

UNIVERSITAT JAUME·I
HPC&A

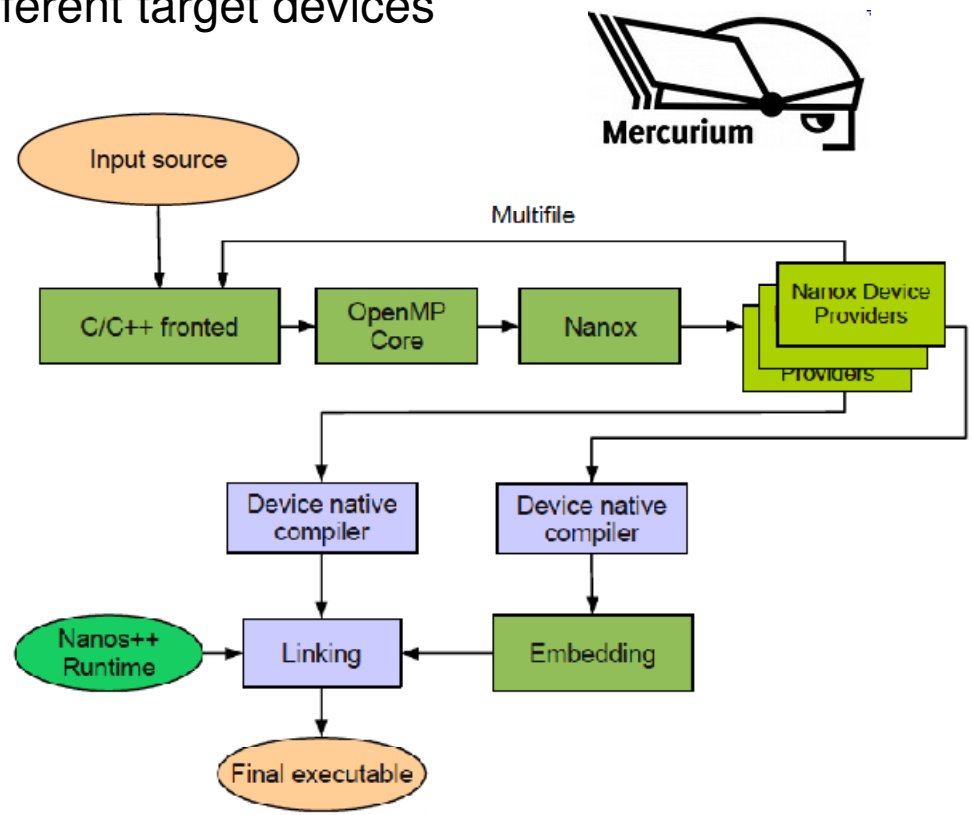
- Nesting

```
#pragma omp task input([BS][BS]A, [BS][BS]B)\
inout([BS][BS]C)
void small_dgemm(float *C, float *A, float *B);

#pragma omp task input([N][N]A, [N][N] B)\
inout([N][N]C)
void block_dgemm(float *C, float *A, float *B){
  int i, j, k;

  for (i=0; i< N; i+=BS)
    for (j=0; j< N; j+=BS)
      for (k=0; k< N; k+=BS)
        small_dgemm(&C[i][j], &A[i][k], &B[k][j])
}

main() {
  ...
  block_dgemm(A,B,C);
  block_dgemm(D,E,F);
  #pragma omp task wait
}
```



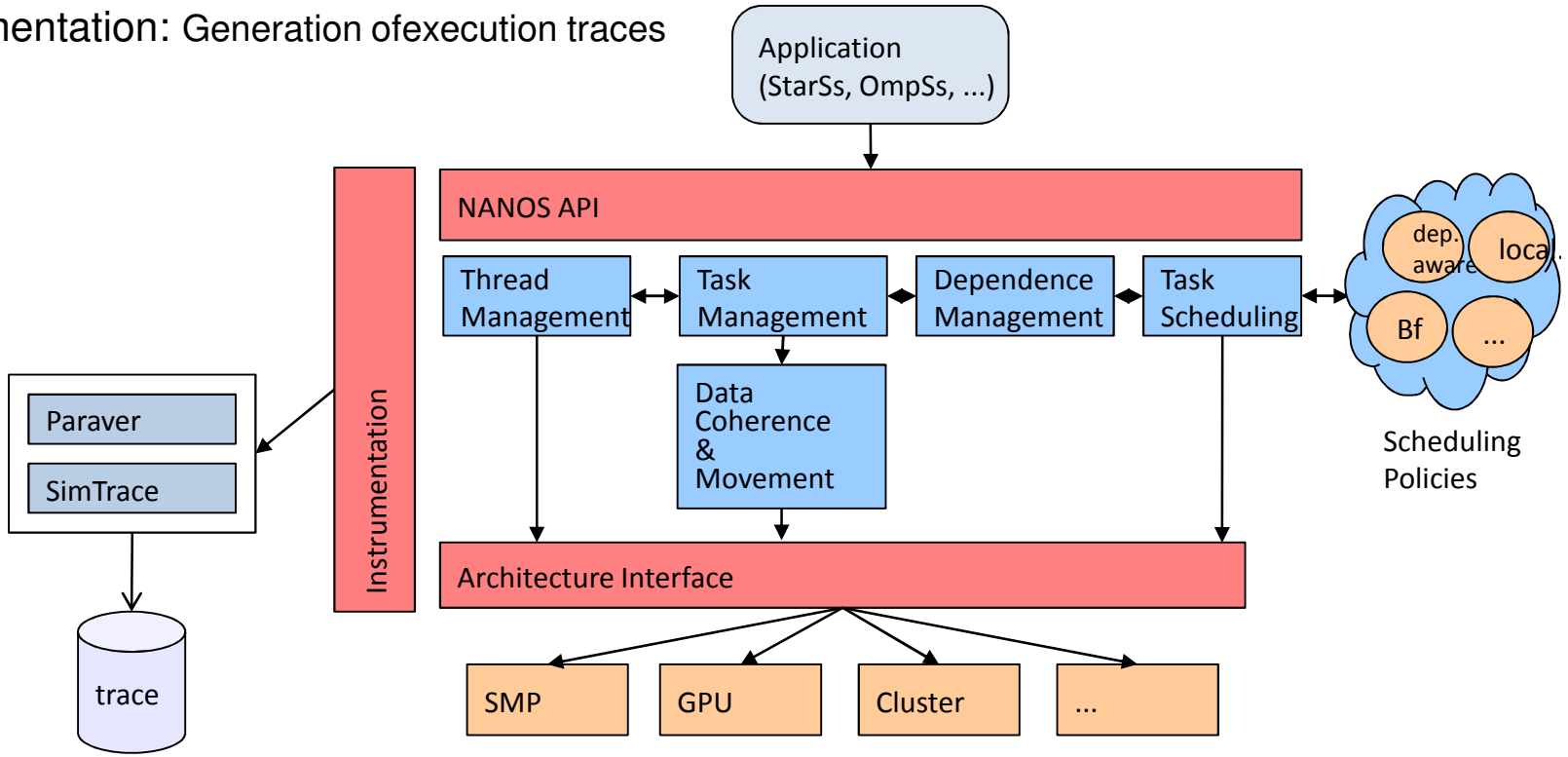
Mercurium

- Minor role
- Recognizes constructs and transforms them to calls to the runtime
- Manages code restructuring for different target devices
  - Device-specific handlers
  - May generate code in a separate file
  - Invokes different back-end compilers
    → nvcc for NVIDIA



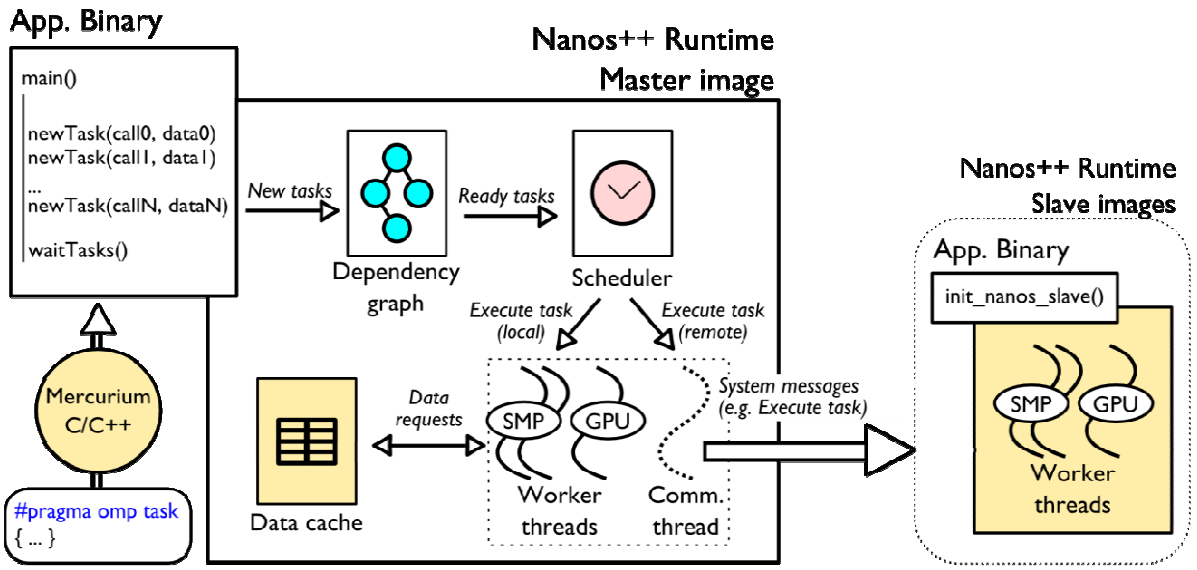
Structure

- Support to different programming models: OpenMP (OmpSs), StarSs, Chapel
- Independent components for thread, task, dependence handling, task scheduling, ...
- Most of the runtime independent of the target arch.: SMP, GPU, simulator, cluster
- Support to heterogeneous targets: i.e., threads running tasks in regular cores and GPUs
- Instrumentation: Generation ofexecution traces



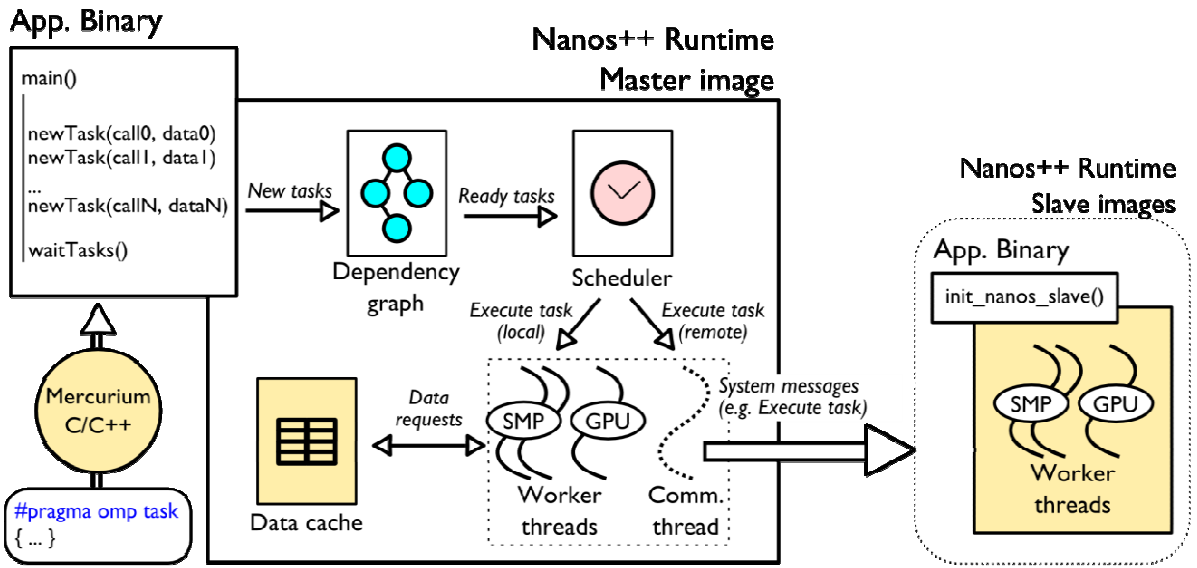
Structure behaviour: task handling

- Task generation
- Data dependence analysis
- Task scheduling



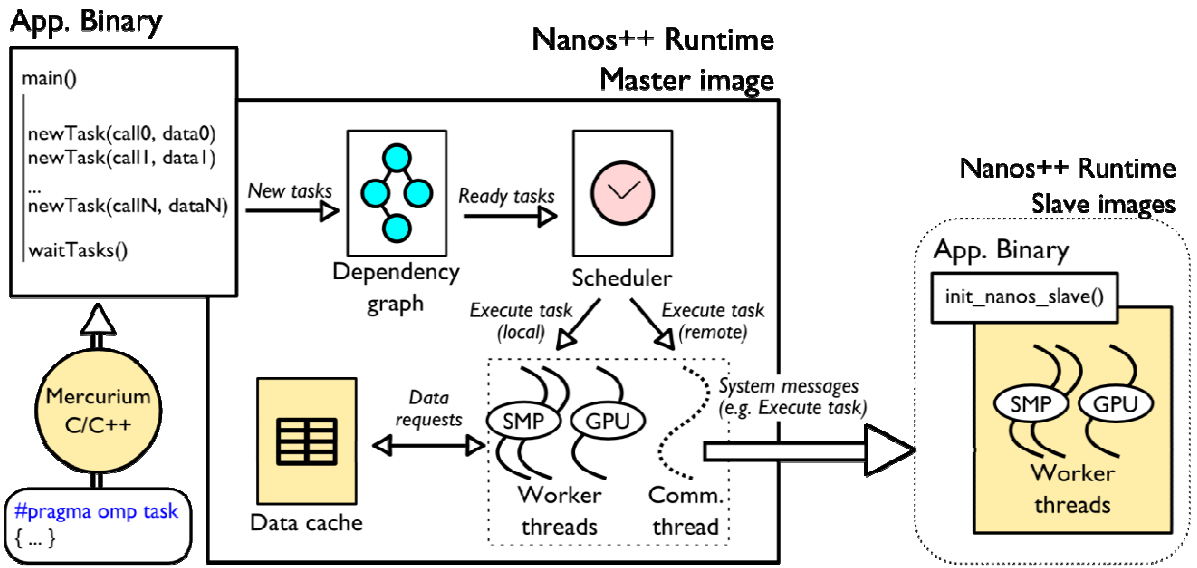
Structure behaviour: coherence support

- Different address spaces managed with:
  - A hierarchical directory
  - A software cache per each:
    - Cluster node
    - GPU
- Data transfers between different memory spaces only when needed
  - Write-through
  - Write-back



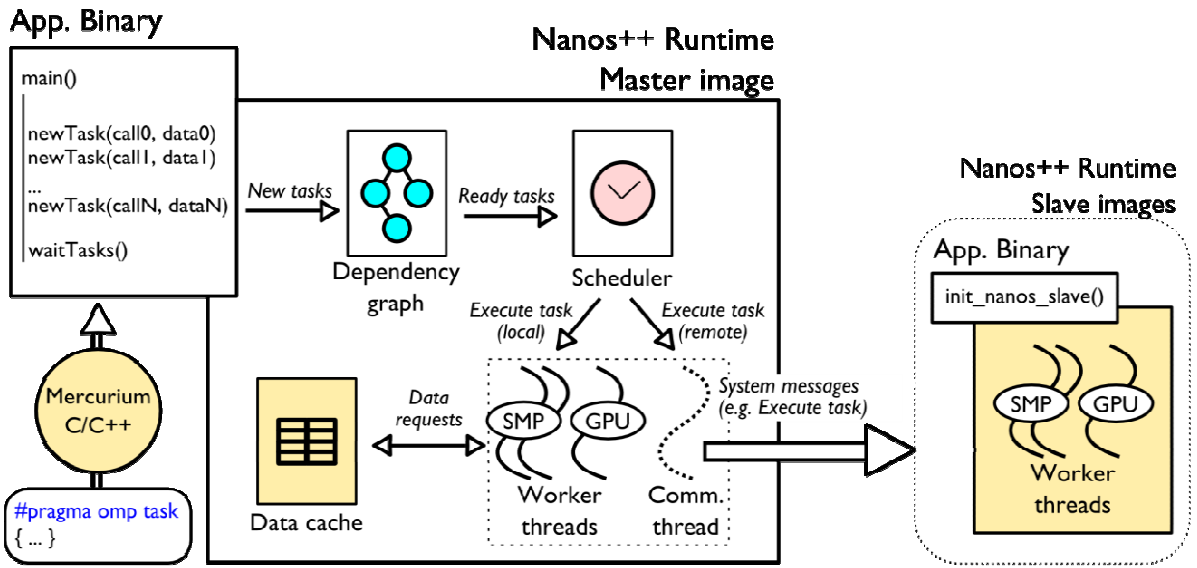
Structure behaviour: clusters

- One runtime instance per node
  - One master image
  - N-1 worker images
- Low level communication through active messages
- Tasks generated by master
  - Tasks executed by worker threads in the master
  - Tasks delegated to slave nodes through the communication thread
- Remote task execution:
  - Data transfer (if necessary)
  - Overlap of computation with communication
  - Task execution
    - Local scheduler



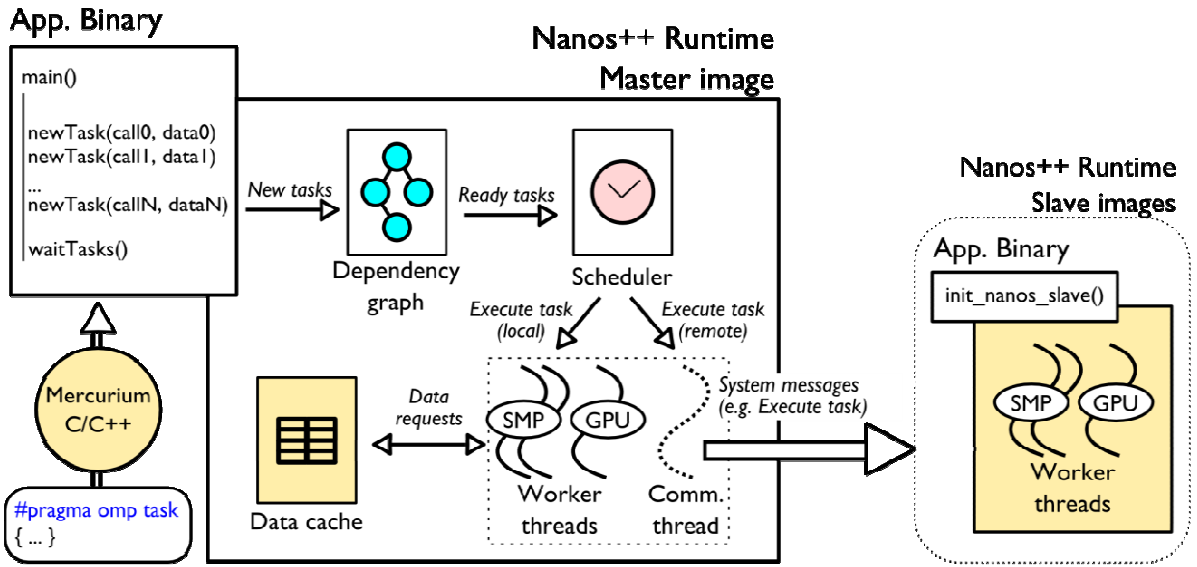
Structure behaviour: GPUs

- Automatic handling of Multi-GPU execution
- Transparent data-managementon GPU side (allocation, transfers, ...) and synchronization
- One manager thread in the host per GPU. Responsible for:
  - Transferring data from/to GPUs
  - Executing GPU tasks
  - Synchronization
- Overlap of computation and communication
- Data pre-fetch



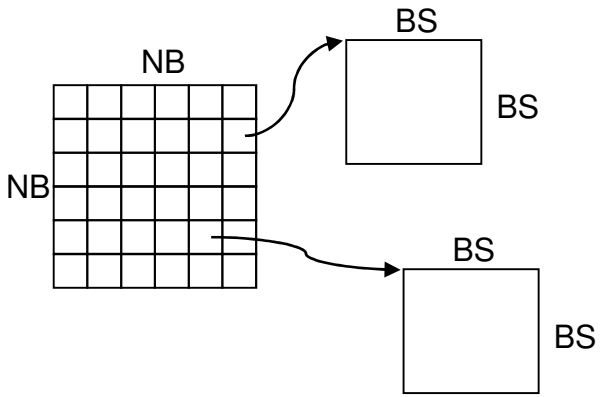
Structure behaviour: clusters of GPUs

- Composes previous approaches
- Supports for heterogeneity and hierarchy:
  - Application with homogeneous tasks: SMP or GPU
  - Applications with heterogeneous tasks: SMP and GPU
  - Applications with hierarchical and heterogeneous tasks:
    - I.e., coarser grain SMP tasks
    - Internally generating GPU tasks



MxM on matrix stored by blocks

```
int main  (int argc, char **argv) {
  int i, j, k;
  ...
  initialize(A, B, C);

  for (i=0; i < NB; i++)
    for (j=0; j < NB; j++)
      for (k=0; k < NB; k++)
        mm_tile( C[i][j], A[i][k],
                 B[k][j]);
}
```



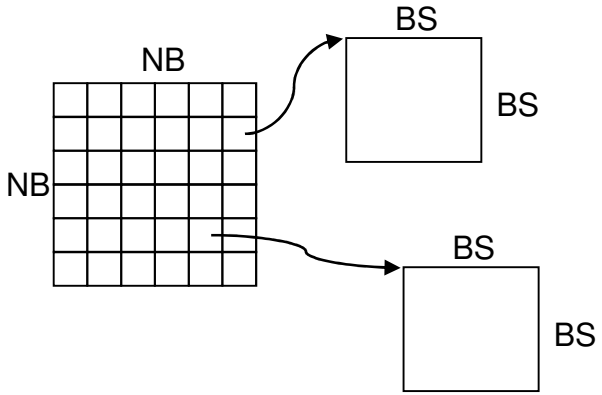
Will work on matrices of any size

Will work on any number of cores/devices

```
#pragma omp task input([BS][BS]A,[BS][BS]B)\
inout([BS][BS]C)
static void mm_tile ( float C[BS][BS],
                      float A[BS][BS],
                      float B[BS][BS]) {
  int i, j, k;

  for (i=0; i< BS; i++)
    for (j=0; j< BS; j++)
      for (k=0; k< BS; k++)
        C[i][j] += A[i][k] * B[k][j];
}
```

MxM on GPUs using CUBLAS kernel

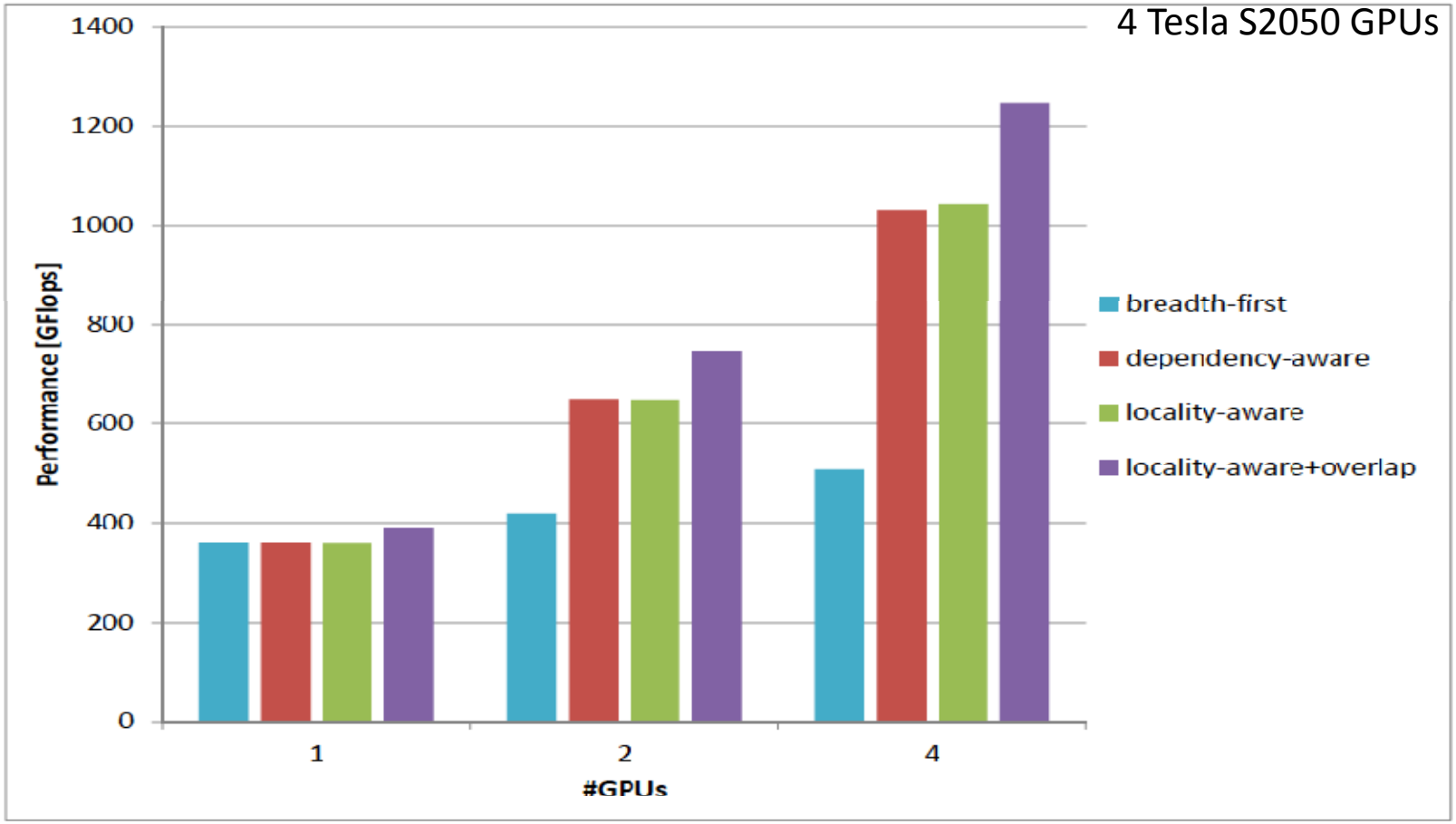
```
int main  (int argc, char **argv) {
  int i, j, k;
  ...
  initialize(A, B, C);

  for (i=0; i < NB; i++)
    for (j=0; j < NB; j++)
      for (k=0; k < NB; k++)
        mm_tile( C[i][j], A[i][k],
                 B[k][j]);
}
```




```
#pragma omp target device (cuda) copy_deps
#pragma omp task input([BS][BS]A, [BS][BS]B, BS)\
inout([BS][BS]C)
void mm_tile(float *A, float *B, float *C, int BS)
{
  unsigned char nt = 'N';
  float sone = 1.0, smone = -1.0;
  float *d_A, *d_B, *d_C;

  cublasSgemm(nt, nt, nt, BS, BS, smone, A,
              BS, B, BS, sone, C, BS);
}
```

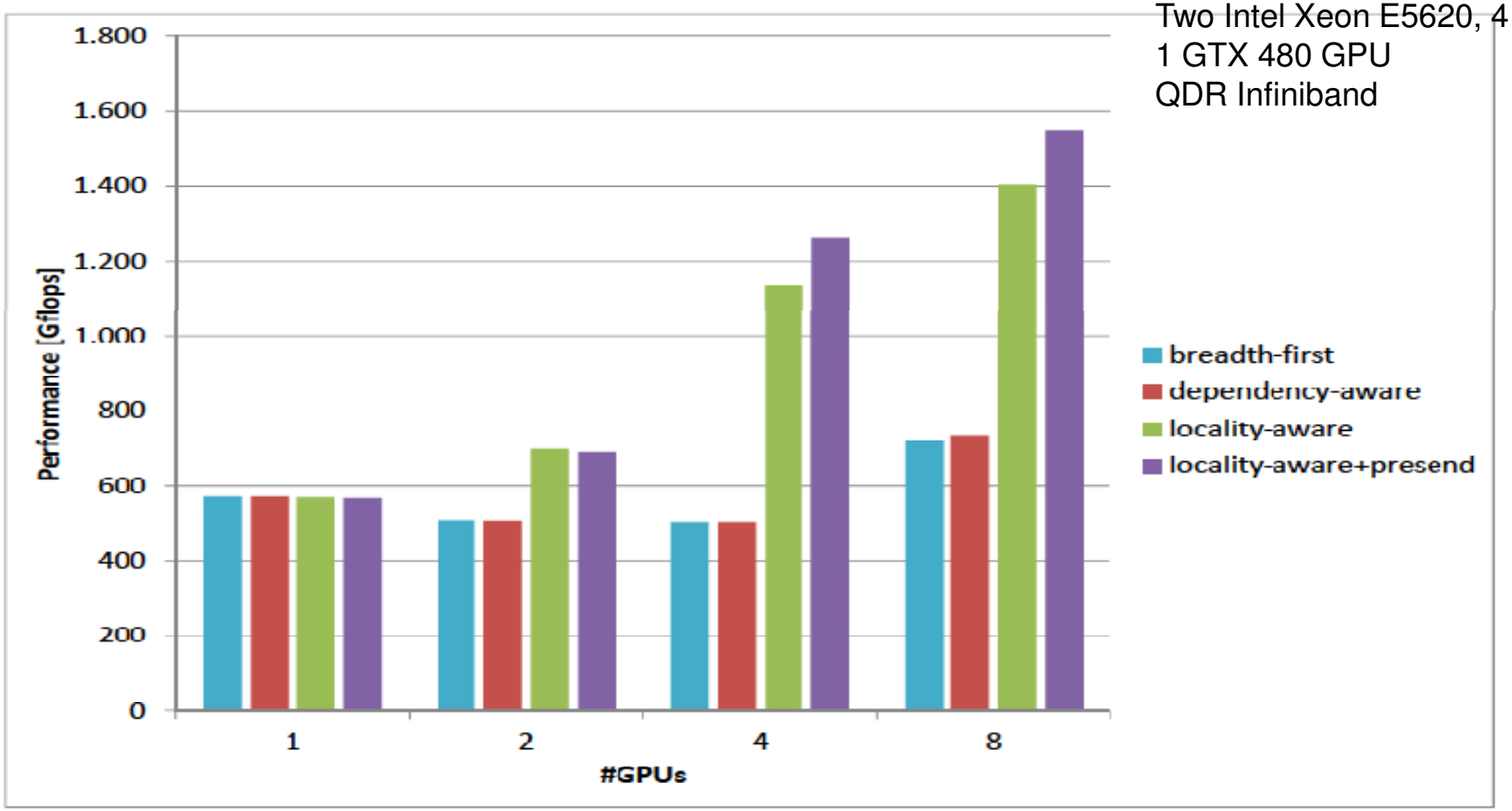
MxM performance on multi-GPU

Two Intel Xeon E5440, 4 cores
4 Tesla S2050 GPUs



MxM performance on cluster of GPUs

8 nodes of DAS-4:
Two Intel Xeon E5620, 4 cores
1 GTX 480 GPU
QDR Infiniband



Cholesky: Block matrix storage on GPU

UNIVERSITAT JAUME·I HPC&A

- Source code independent of # devices

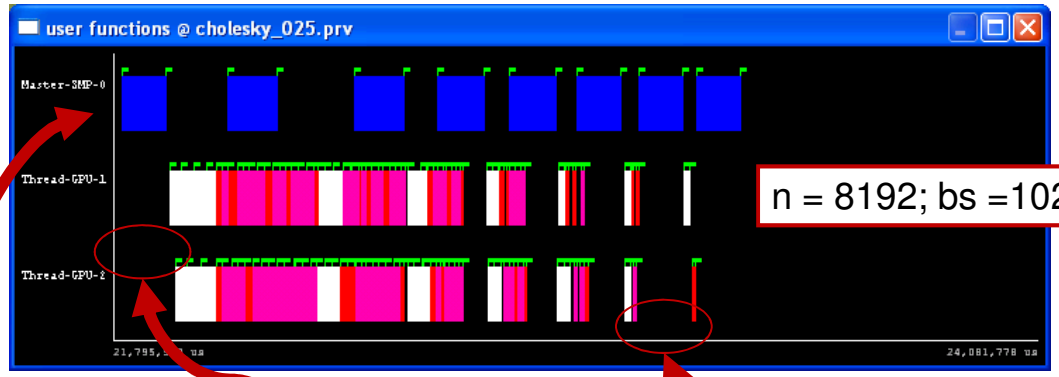

```
void blocked_cholesky( int NB, float *A
  int i, j, k;
  for (k=0; k<NB; k++) {
    spotrf (A[k*NB+k]);
    for (i=k+1; i<NB; i++)
      strsm (A[k*NB+k], A[k*NB+i]);
      for (i=k+1; i<NB; i++) {
        for (j=k+1; j<i; j++)
          sgemm( A[k*NB+
        ssyrk (A[k*NB+i]
      }
  }
  #pragma omp task wait
}
```

Spotrf:
Slow task @ GPU
In critical path (scheduling problem)

```
#pragma omp target device (cuda) copy_deps
#pragma omp task inout([BS][BS]A)
void spotrf (float *A);
#pragma omp target device (cuda) copy_deps
#pragma omp task input ([BS][BS]A) inout([BS][BS]C)
void ssyrk (float *A, float *C);
#pragma omp target device (cuda) copy_deps
#pragma omp task input ([BS][BS]A, [BS][BS]B) inout([BS][BS]C)
void sgemm (float *A, float *B, float *C);
#pragma omp target device (cuda) copy_deps
#pragma omp task input ([BS][BS]T) inout([BS][BS]B)
void strsm (float *T, float *B);
```

Cholesky: Heterogeneous execution

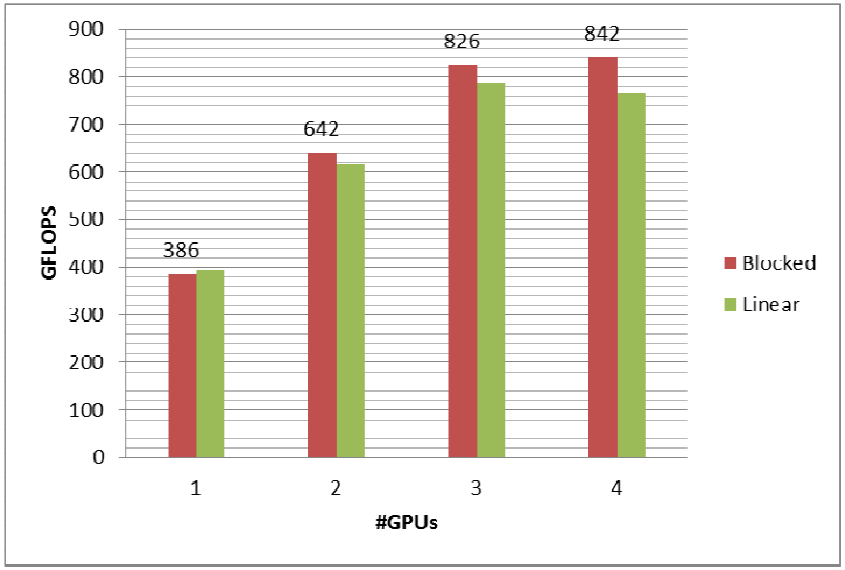
- `spotrf` more efficient at CPU
- Overlap between CPU and GPU

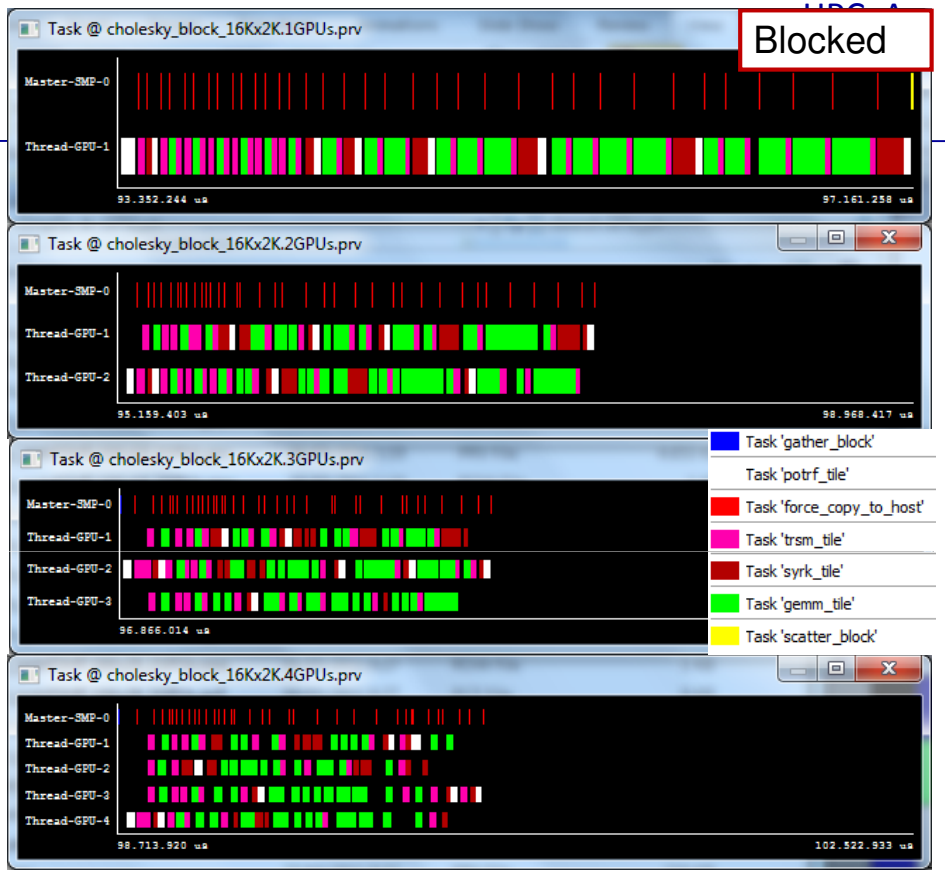


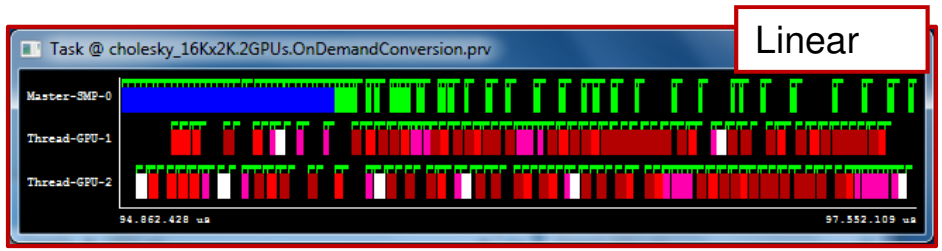

```
#pragma omp target device (smp) copy_deps
#pragma omp task inout([BS][BS]A)
void spotrf_tile(float *A, int BS)
{
  long INFO;
  char L = 'L';

  spotrf_( &L, &BS, A, &BS, &INFO );
}
```

Cholesky performance

- Matrixsize: 16K x 16K
- Block size: 2K x 2K
- Storage: Blocked / contiguous
- Tasks:
  - spotrf: MAGMA
  - trsm, syrk, gemm: CUBLAS







Further information

UNIVERSITAT JAUME·I
HPC&A

- Contact:
  - rosa.m.badia@bsc.es
- Barcelona Supercomputing Center:
  - www.bsc.es

Farewell

UNIVERSITAT JAUME·I
HPC&A

Thanks for your attention!*

*Hope you enjoyed this as much as Antibes-Sophia's beach