SuperMatrix Out-of-Order Scheduling of Matrix Operations for SMP and Multi-core Architectures

Enrique S. Quintana-Ortí

UNIVERSITAT JAUME I

Dept. de Ingeniería y Ciencia de Computadores
Universidad Jaime I de Castellón
quintana@icc.uji.es

Developing dense LA libraries

- FLAME: joint project with Robert van de Geijn (UT-Austin)
  http://www.cs.utexas.edu/users/flame

  - Methodology for formal derivation of dense LA algorithms
  - Optimization on HPC architectures

- Support from:
  - NSF CCF-0540926 “Foundations of programming linear algebra algorithms on SMP and multicore systems”
  - NSF CCF-0702714 “Foundations and applications of hierarchically stored matrices”

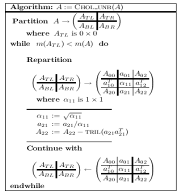
Outline

- Motivation
- FLAME
- FLAME for SMP and multi-core architectures

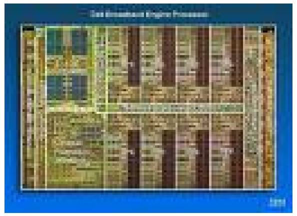
Outline

- Motivation

  Developing dense LA libraries for HPC architectures

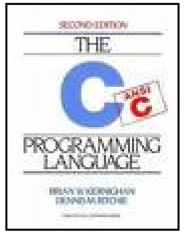


Algorithms



Architectures (hardware)

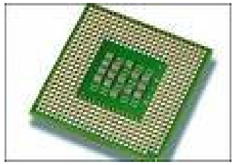
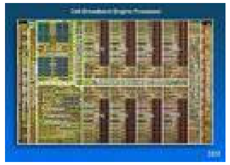
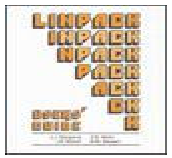
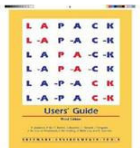
Compilers, languages, and libraries



- FLAME
- FLAME for SMP and multi-core architectures
- Ongoing research

Motivation

- Development of dense LA libraries:
  - Needs of scientists and engineers
  - “Current” HPC architecture

1975-	1985-	1995-	2005-	2015-
Vector	µproc.	Clusters	Multi-core,FPGA, GPU	¿?
				
BLAS-1, LINPACK, EISPACK	BLAS-2, BLAS-3, LAPACK	ScaLAPACK, PLAPACK	¿?	¿?
				

How to develop the final dense LA library?

Motivation

- What features should one expect from a final library?

Functionality, high performance, portability, and accuracy of current libraries

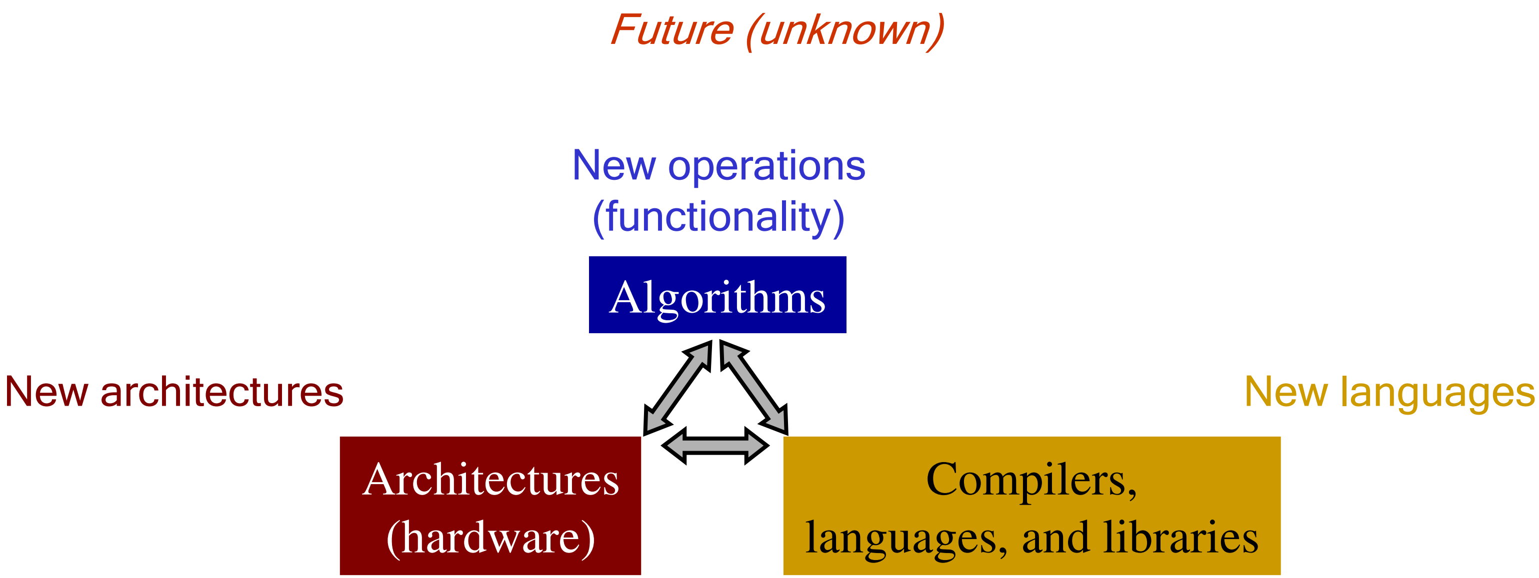
The real challenge is compatibility with the *future (unknown)*

1975-	1985-	1995-	2005-	2015-
Vector	µproc.	Clusters	Multi-core, FPGA, GPU	¿?
BLAS-1, LINPACK, EISPACK	BLAS-2, BLAS-3, LAPACK	ScaLAPACK, PLAPACK	¿?	¿?

Motivation

- What features should one expect from a final library?

Functionality, high performance, portability, and accuracy of current libraries



Motivation

- How to develop the final dense LA library?

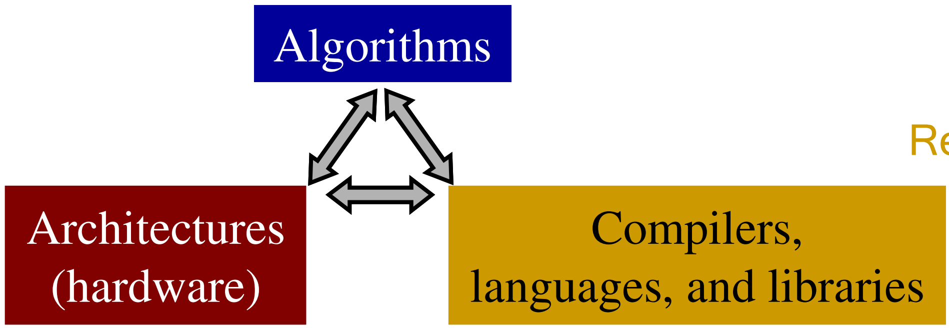
Mid-term goal of FLAME

Tools for the
mechanic (automatic) generation from:

Mathematical specification of the operation

Model of the architecture

Rewritting rules of the language

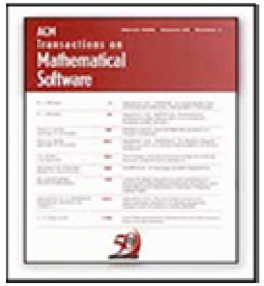


Outline

- Motivation
- FLAME

  [ACM1] P. Bientinesi, J. A. Gunnels, M. Myers, E. S. Quintana, R. van de Geijn, *“The science of deriving dense linear algebra algorithms”*, ACM TOMS, 2005

  → µprocessors

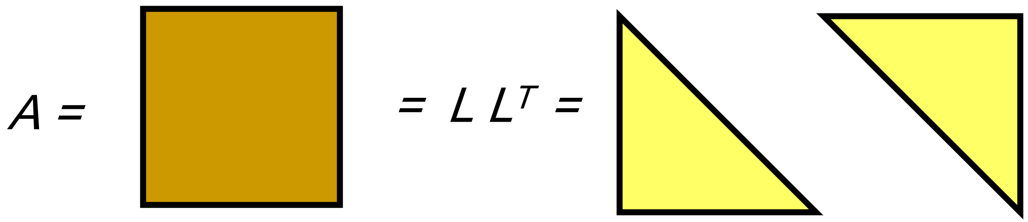


  - Notation
  - New operations
  - New languages
  - New architectures

- FLAME for SMP and multi-core architectures

FLAME

- Case study: Cholesky factorization of (s.p.d.) matrix *A*



*Needed in the solution of a certain class of linear systems $A\,x = b$

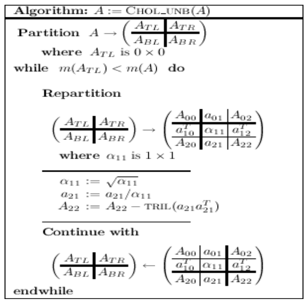
FLAME: systematic derivation

Scientific method

Problem

Experience & art

Algorithm



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

[ACM1]: *Systematic procedure* for dense LA
- Composed of 8 steps

- The first two steps determine the following ones:

- 1st step: precondition and poscondition

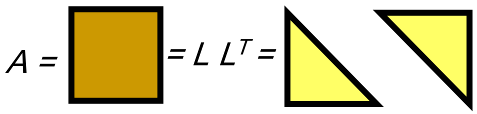
$$P_{pre} : \ (\hat{A} = \hat{A}^T) \ \& \ (\hat{A} > 0)$$
$$P_{pos} : \ (\text{TRIL}(A) = L) \ \& \ (LL^T = \hat{A})$$

- 2nd step: loop-invariant

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c|c} L_{TL} & A_{TR} \\ \hline L_{BL} & A_{BR} - L_{BL} L_{BL}^T \end{array}\right)$$

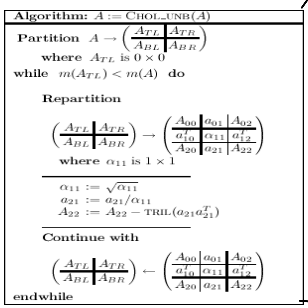
FLAME: systematic derivation

Problem



Systematic procedure

Algorithm



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

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

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: systematic derivation

- Current status:

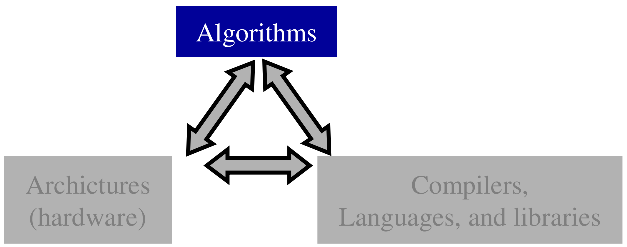
Applied with success to BLAS-1, BLAS-2, BLAS-3, and a major part of LAPACK

- Impact:

Systematic derivation of dense LA from mathematical specifications of the operations

New operations (functionality)

New architectures



New languages

FLAME: application programming interfaces

Algorithm

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

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

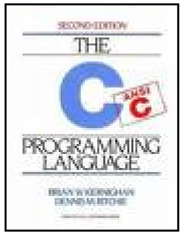
Continue with

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

endwhile

Translation?

Code



[ACM1]: C interface

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

...

```
int Chol_unb( FLA_Obj A )
{
  ...
  FLA_Part_2x2( A,    &ATL, &ATR,
                      &ABL, &ABR,    0,  0, FLA_TL );
  ...
}
```

FLAME: application programming interfaces

- Current status:

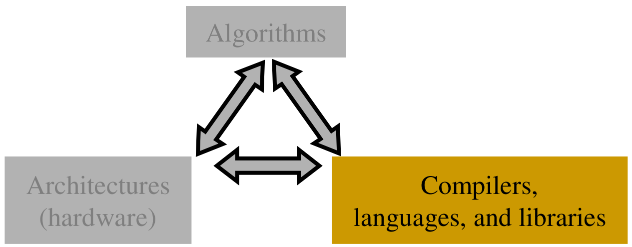
Algorithm → APIs → C, Fortran / Matlab, Labview / LaTeX

- Impact:

The knowledge (algorithm) remains unchanged when a new language appears; it suffices to develop the corresponding *API*

New operations (functionality)

New architectures



New languages

FLAME: families of algorithms

Algorithm

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

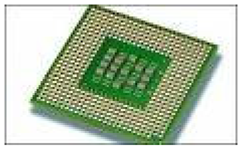
  $\alpha_{11} := \sqrt{\alpha_{11}}$
  $a_{21} := a_{21} / \alpha_{11}$
  $A_{22} := A_{22} - \text{TRIL}(a_{21} a_{21}^T)$

  Continue with

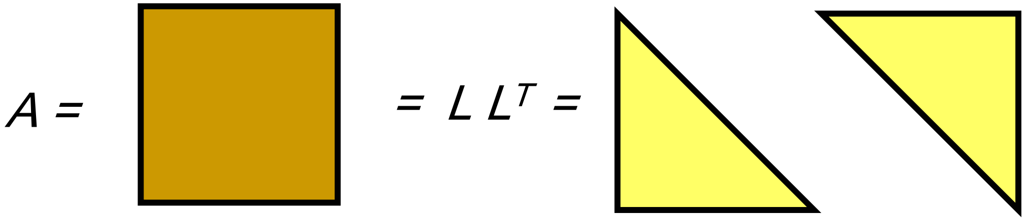
  $\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$
endwhile

Optimization?

Architecture



For each operation, there exist several algorithms



3 scalar *variants* and
3 blocked *variants*:
families of algorithms

FLAME: families of algorithms

Algorithm

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

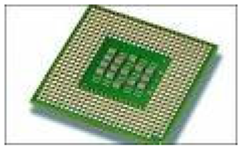
Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$
  where A_{TL} is 0×0
while $m(A_{TL}) < m(A)$ **do**
  Repartition
  $$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$$
  where α_{11} is 1×1

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

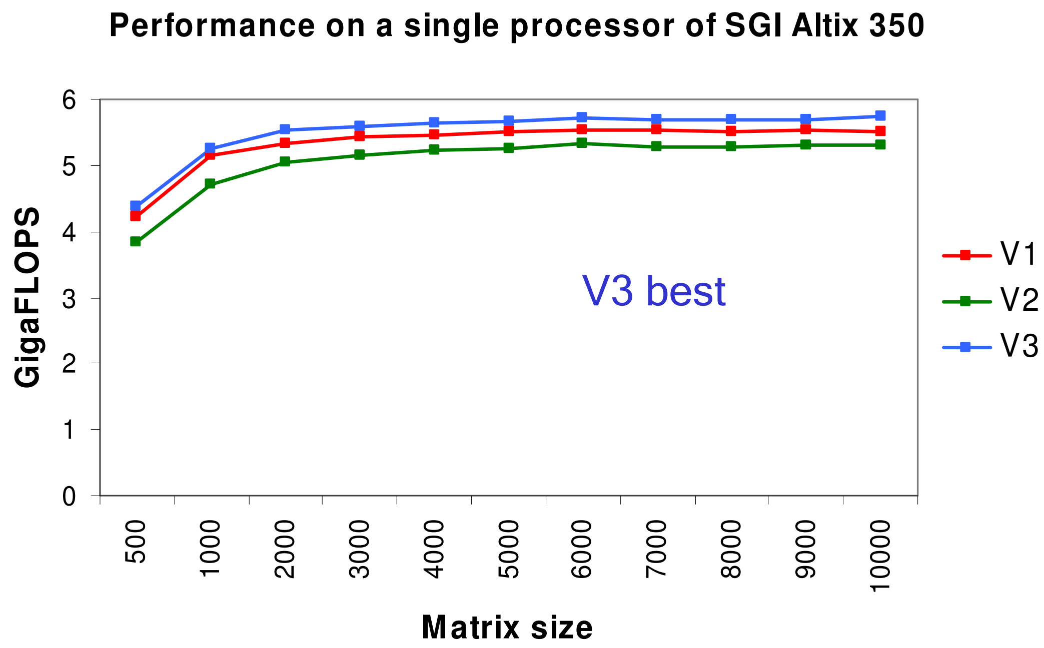
  Continue with
  $$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$$
endwhile

Optimization

Architecture



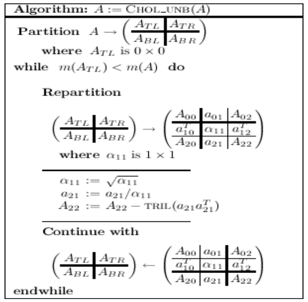
Different variants yield different performance



Itanium2@1.5GHz (IA-64) L3 6MB

FLAME: families of algorithms

Algorithm


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

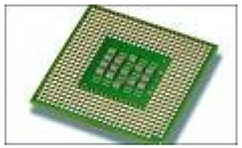
Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$
where A_{TL} is 0×0
while $m(A_{TL}) < m(A)$ do
Repartition
$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$
where α_{11} is 1×1

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

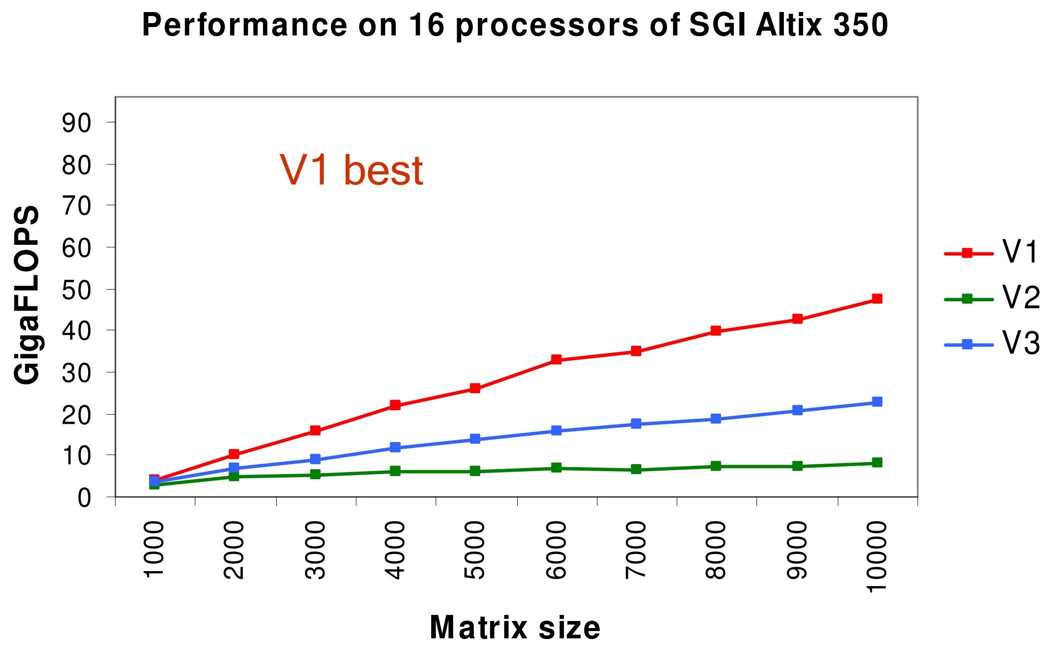
Continue with
$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$
endwhile

Optimization

Architecture



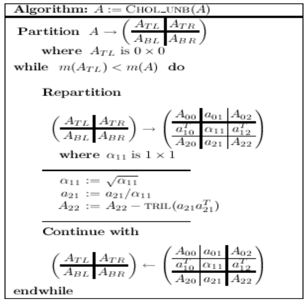
Also on SMPs



16 x Itanium2@1.5GHz (IA-64) L3 6MB
+ SGI NUMAlink

FLAME: families of algorithms

Algorithm


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

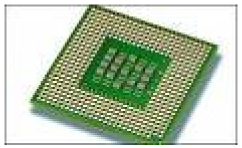
Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$
  where A_{TL} is 0×0
while $m(A_{TL}) < m(A)$ **do**
  Repartition
  $\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$
   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

Optimization

Architecture



[ACM1]: Generation of families of algorithms

Precondition, poscondition,
and loop-invariants $I_1, I_2, \ldots, I_n$

Automatic generation

Algorithms $V_1, V_2, \ldots, V_n$

FLAME: families of algorithms

Algorithm

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

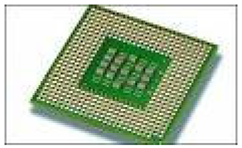
Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$
where A_{TL} is 0×0
while $m(A_{TL}) < m(A)$ **do**
Repartition
$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$$
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

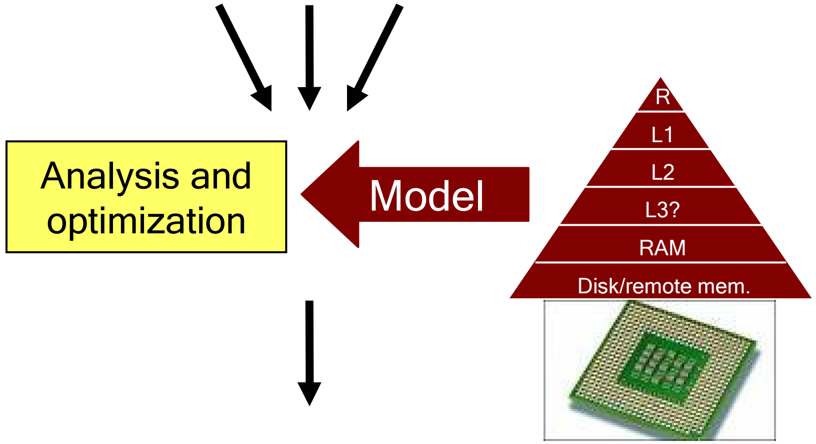
Optimization

Architecture



[ACM1]: Analysis of families of algorithms

Algorithms $V_1, V_2, \ldots, V_n$



Optimal performance

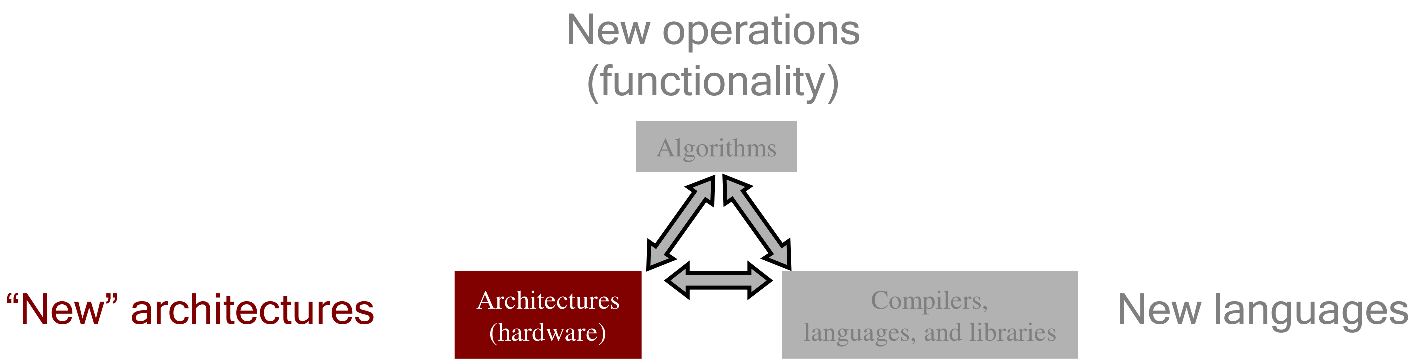
FLAME: families of algorithms

- Current status: libFLAME 0.9

Applied with success to BLAS-1, BLAS-2, BLAS-3, and a major part of LAPACK, for IA-32, IA-64 and SMPs

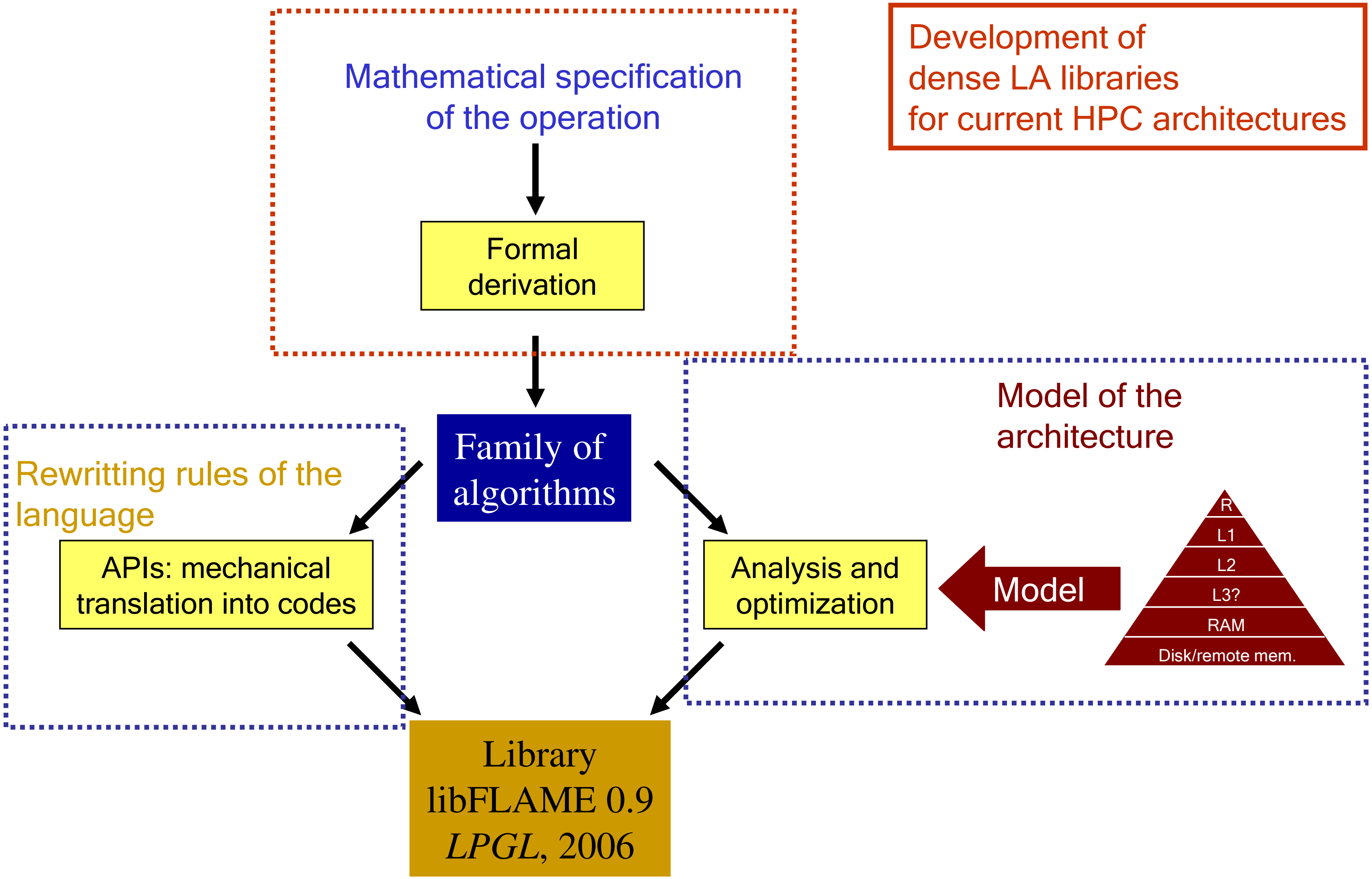
- Impact:

Exploring the best option among several (algorithms) for a specific architecture



FLAME:
-Notation
-New operations
-New languages
-New architectures

FLAME: summary



Outline

- Motivation
- FLAME
- FLAME for SMP and multi-core architectures

  [ACM2] E. Chan, E. S. Quintana, G. Quintana, R. van de Geijn, *"Supermatrix out-of-order scheduling of matrix operations for SMP and multi-core architectures"*, 19th ACM SPAA, 2007

  → SMP and multi-core

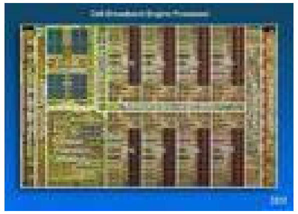
  - Motivation
  - Improving the scalability
  - Improving the locality of reference
  - Results

FLAME multi-core: motivation

- FLAME is forward compatible to *future* architectures!

  Multi-core processors or CMPs (*chip multiprocessors*)

  -IBM+Sony+Toshiba CELL BE: 1+8 cores



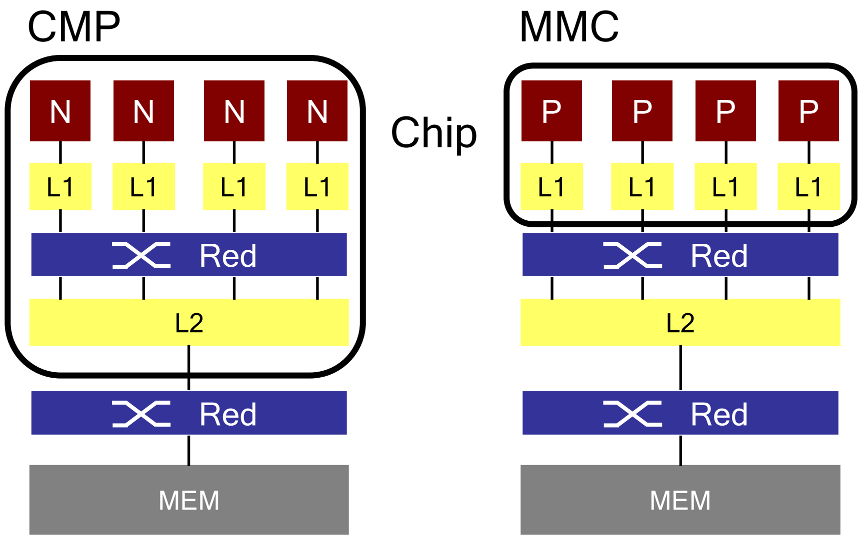
  - SUN UltraSparc Niagara T1 (8 cores), Intel Quad Core (4 cores), AMD Athlon 64 x2 (2 cores)

  Future?

  - Intel prototype with 80 cores, manycore in the near future
  - Double #cores per generation

FLAME multi-core: motivation

- CMP ≠ Shared Memory Multiprocessors (SMPs)
  - Scale: hundreds of cores in a chip
  - Heterogeneity in CMPs (e.g., systems with cores with different capabilities)
  - Organization



Network in chip: fast/cheap communications between cores in CMPs

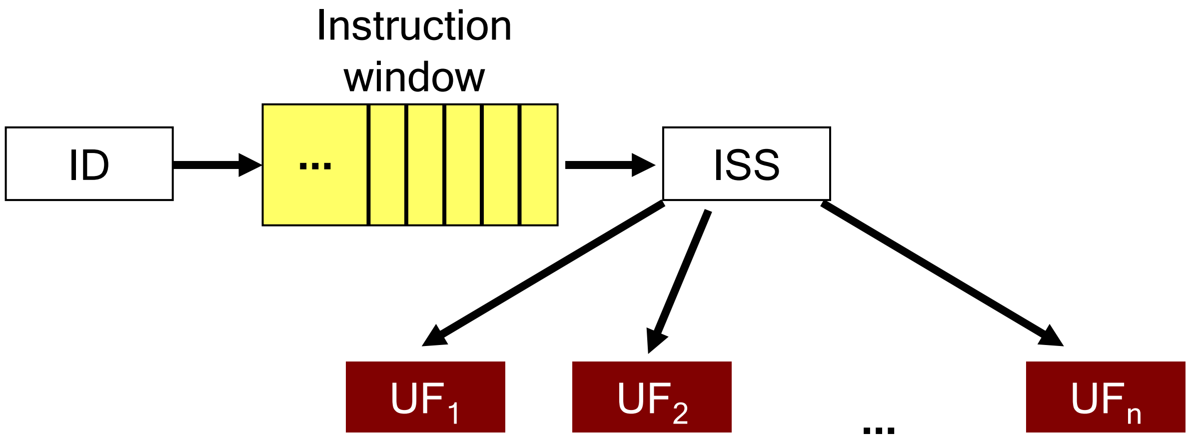
FLAME multi-core: motivation

- Requirements on libraries for CMPs:
  - Scalability
  - Flexibility (for heterogeneity)
  - Locality of reference (keep communications inside chip)

- Where are we now (SMPs)?
  - Artificial limits to the degree of concurrency
  - Implementation only for experts
  - Locality of reference based solely on blocked algorithms

FLAME multi-core: scalability

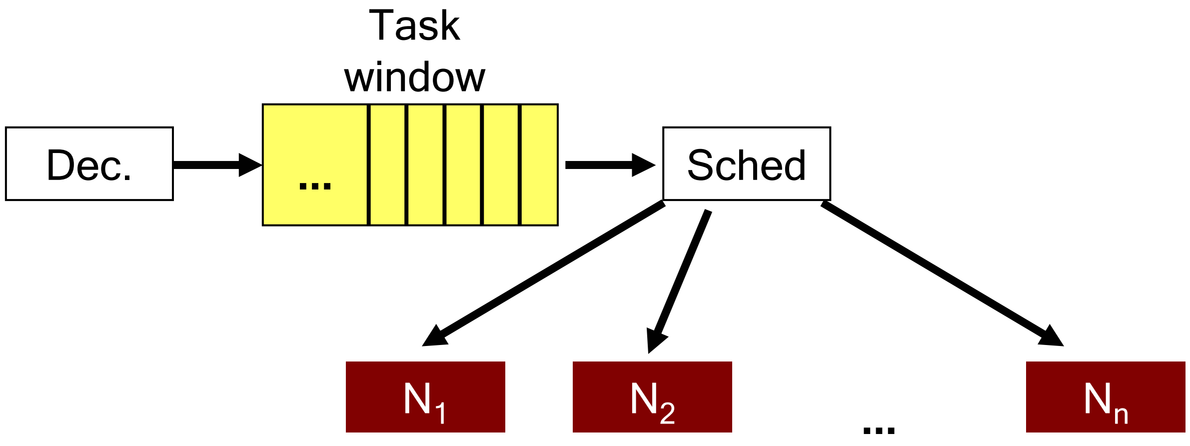
- Organization of a superscalar processor:



1. In-order decodification stage (*Instruction Level Parallelism*)
2. Out-of-order issue stage, preserving dependences (Tomasulo)
3. Parallel execution

FLAME multi-core: scalability

- Organization of *scalable parallel processing* in CMP (proposal)



1. Decomposition into task in the first stage of execution (*Data/Task Level Parallelism*)
2. Dynamic scheduling
3. Out-of-order issue stage, preserving dependences (Tomasulo)

FLAME multi-core: scalability

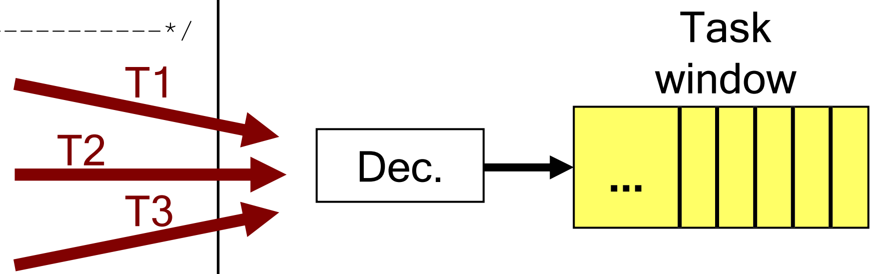
1. Decomposition into tasks (automatic stage)

libFLAME

```
int Chol_blk( FLA_Obj A, int b)
{
  ...
  while ( FLA_Obj_length( ATL ) < FLA_Obj_length( A ) ){
    ...
    /* ------------------------------------------------*/
    /* A11 := Chol_unb( A11 ) */
    Chol_unb( A11 );

    /* A21 := A21 * TRIL( A11 )^-T */
    FLA_Trsm( ..., A11, A21 );

    /* A22 := A22 - TRIL( A21 * A21^T ) */
    FLA_Syrk( ..., A21,..., A22 );
/* --------------------------------------------------------*/
 ...
}
```



- Decomposition module is common to all library
- Recursive decomposition; 2-D for scalability
- Task size: *b*

FLAME multi-core: scalability

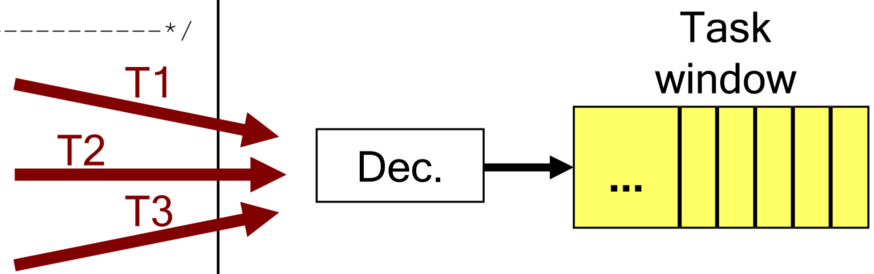
2. Dynamic scheduling = *Spatial* assignment of tasks to cores

```
int Chol_blk( FLA_Obj A, int b)
{
  ...
  while ( FLA_Obj_length( ATL ) < FLA_Obj_length( A ) ){
    ...
    /* ------------------------------------------------------*/
    /* A11 := Chol_unb( A11 ) */
    Chol_unb( A11 );

    /* A21 := A21 * TRIL( A11 )^-T */
    FLA_Trsm( ..., A11, A21 );

    /* A22 := A22 - TRIL( A21 * A21^T ) */
    FLA_Syrk( ..., A21,..., A22 );
/* ----------------------------------------------------------*/
 ...
}
```

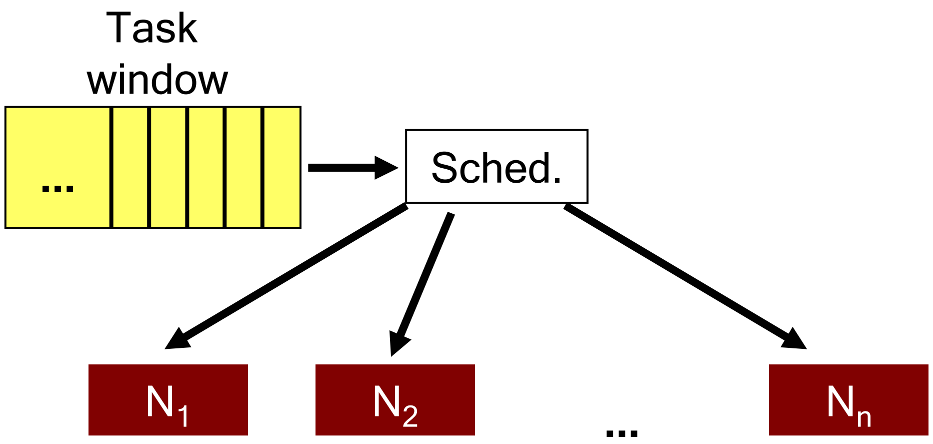
libFLAME



- Different computational cost of tasks requires dynamic scheduling for balancing

FLAME multi-core: scalability

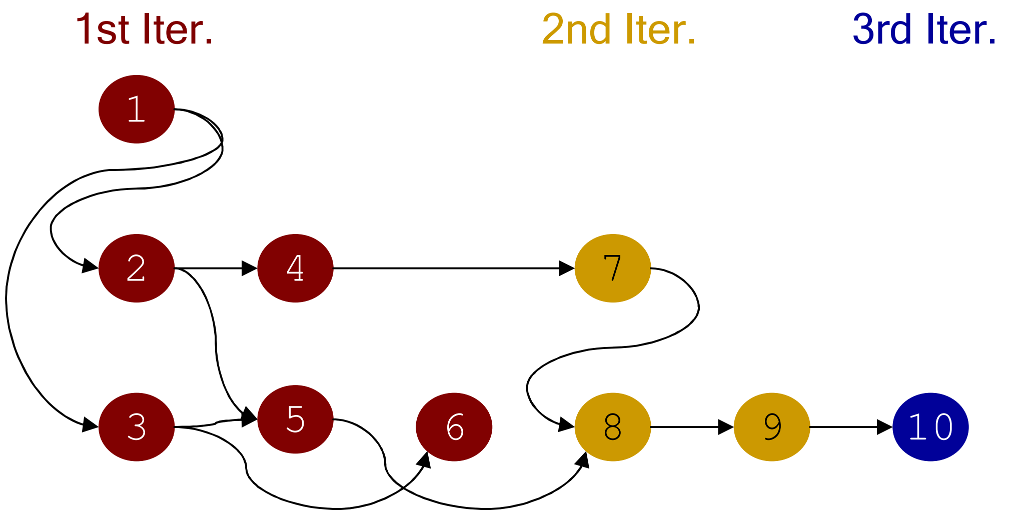
3. Out-of-order issue = *Temporal* assignment of tasks to cores



- Scheduler (module) common to all library; architecture-aware
- Schedule first those tasks in the *critical path*

FLAME multi-core: scalability

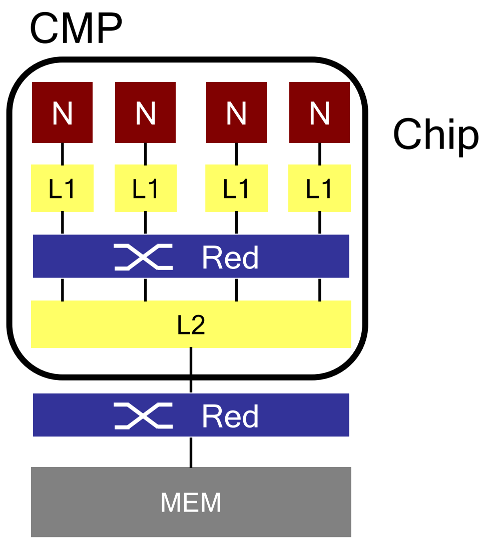
3. Out-of-order issue



- Concurrency is only limited by data dependencies: *data/task parallelism*
- Keeping track of dependencies: software implementation of Tomasulo's algorithm

FLAME multi-core: locality of reference

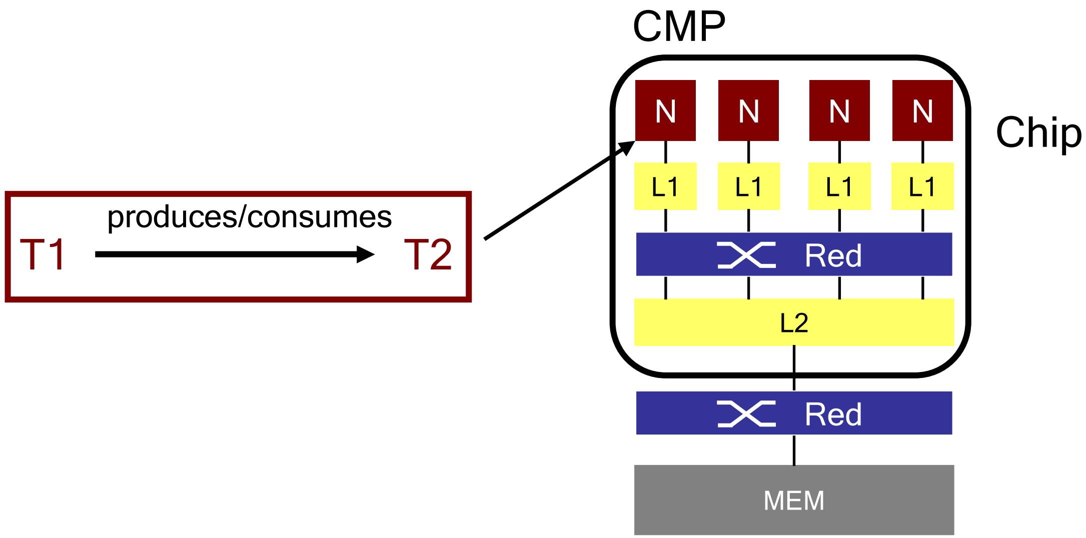
- Reducing off-chip communications



1. Affinity of tasks, threads, and cores
2. Recursive storage for matrices

FLAME multi-core: locality of reference

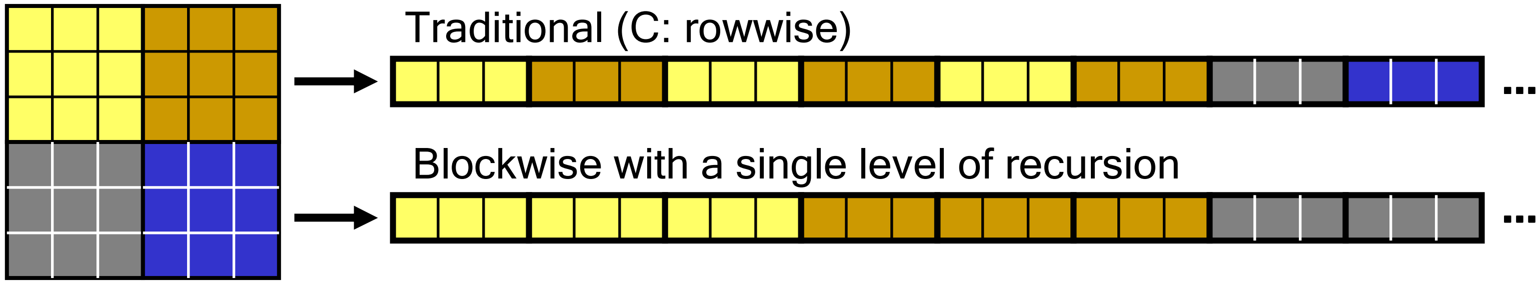
1. Affinity of tasks, threads, and cores



- *A thread per core*, which hopefully does not migrate during exectution (depends on O.S.)
- Dynamic scheduling of tasks to threads yes, but *not random*:
  locality of reference to local caches

FLAME multi-core: locality of reference

2. Recursive storage for matrices

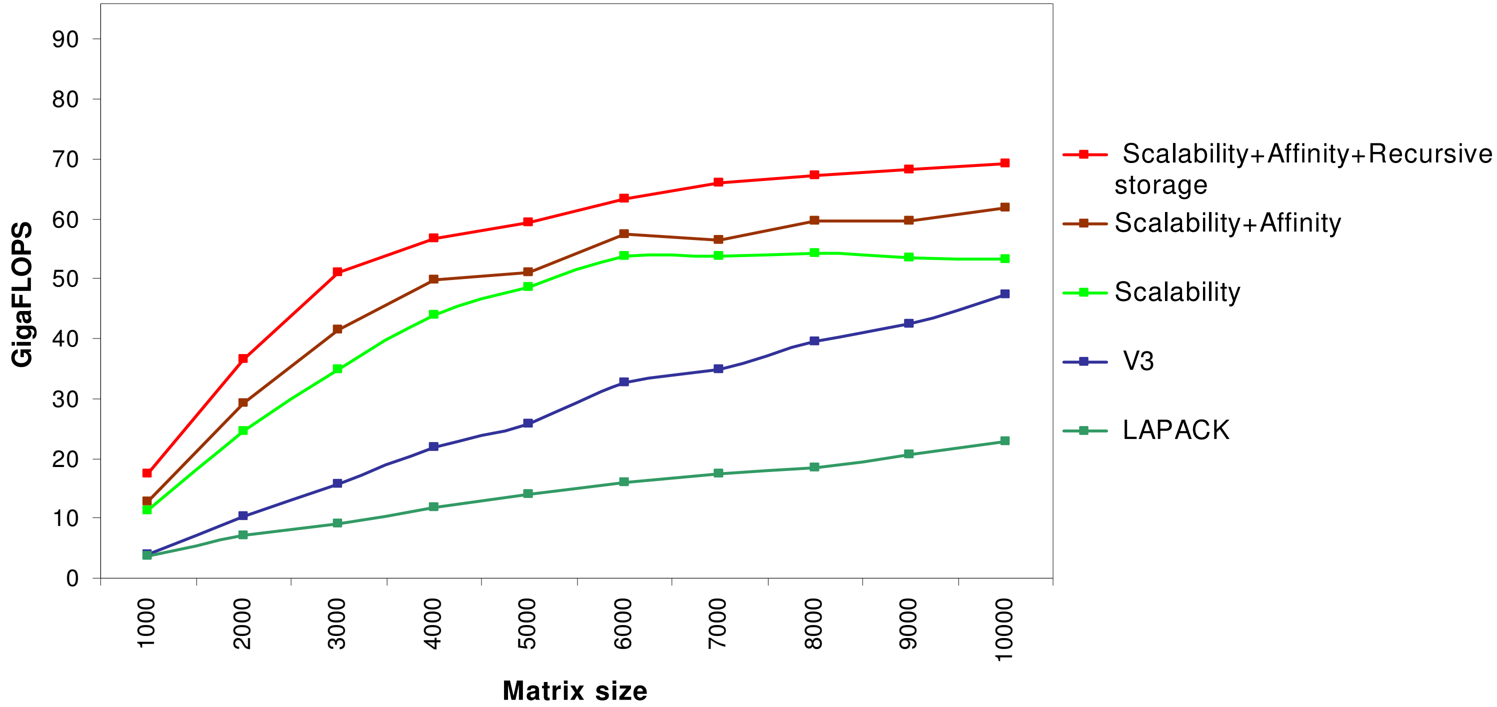


- No need to pack/unpack blocks
- Operating by blocks reduces the number of data/TLB caches misses

- On the other hand, the storage schemes becomes less intuitive:
  → FLASH API

FLAME multi-core: results

- Preliminary: SMP ≈ CMP



16 x Itanium2@1.5GHz (IA-64) L3 6MB

FLAME multi-core: summary

- Current status:

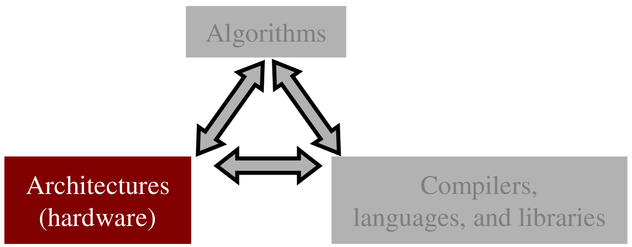
Applied with success to Cholesky and LU w/out pivoting factorizations, BLAS-3;
LU with pivoting and QR factorization under development

- Impact:

The knowledge (library) remains unchanged; it suffices to develop the corresponding runtime system (dec.+scheduler)

New operations (functionality)

New architectures



New languages

FLAME: summary

Development of
dense LA libraries
for multi-core processors