Enabling Partial Pivoting in Task Flow LU Factorization

Master Defense

Omar ZENATI

Supervisors: G. Bosilca, P. Ramet

September 13, 2012

- Three months at *Innovative Computer Laboratory*

- Three months at *Inria Bordeaux Sud-Ouest*

Summary

Context and Motivations

- $A = LU$
- $PA = LU$

LU Decomposition Without Pivoting ($A = LU$)

LU Decomposition with Partial Pivoting ($PA = LU$)

- Panel Factorization
- Update of Trailing Sub-matrix

Performance

Summary

Context and Motivations

$A = LU$

$PA = LU$

LU Decomposition Without Pivoting ($A = LU$)

LU Decomposition with Partial Pivoting ($PA = LU$)

Panel Factorization

Update of Trailing Sub-matrix

Performance

Architecture Trends

- Computing platforms are more and more complex
- We classify them into four categories:
  - Shared memory architectures
  - Distributed memory architectures
  - Hierarchical architectures
  - Heterogeneous architectures

Programming Paradigm

Historically, we use:

- Message passing (MPI) for distributed memory architectures
- Threads library (OMP) for shared memory architectures
- Accelerator library (OpenCL) for accelerator of heterogeneous architectures

$\Rightarrow$ Several programs for the same algorithm
$\Rightarrow$ Weak portability
$\Rightarrow$ Weak scalability

Solution:

Recent paradigm of programming which separate algorithm from architectures used: **Task Flow Model**

Programming Paradigm

Historically, we use:

- Message passing (MPI) for distributed memory architectures
- Threads library (OMP) for shared memory architectures
- Accelerator library (OpenCL) for accelerator of heterogeneous architectures

$\Rightarrow$ Several programs for the same algorithm
$\Rightarrow$ Weak portability
$\Rightarrow$ Weak scalability

Solution:

Recent paradigm of programming which separate algorithm from architectures used: **Task Flow Model**

Programming Paradigm

Historically, we use:

- Message passing (MPI) for distributed memory architectures
- Threads library (OMP) for shared memory architectures
- Accelerator library (OpenCL) for accelerator of heterogeneous architectures

⇒ Several programs for the same algorithm
⇒ Weak portability
⇒ Weak scalability

Solution:

Recent paradigm of programming which separate algorithm from architectures used: **Task Flow Model**

Task Flow Model

Programs can be represented by a Direct Acyclic Graph (DAG) where:

- Vertices are tasks
- Edges are data dependencies between tasks

⇒ One single implementation of the DAG in a specific language

Execution of tasks and data movement between them assured by the **Runtime**

Task Flow Model

Programs can be represented by a Direct Acyclic Graph (DAG) where:

- Vertices are tasks
- Edges are data dependencies between tasks

$\Rightarrow$ One single implementation of the DAG in a specific language

Execution of tasks and data movement between them assured by the **Runtime**

Task Flow Model

Programs can be represented by a Direct Acyclic Graph (DAG) where:

- Vertices are tasks
- Edges are data dependencies between tasks

⇒ One single implementation of the DAG in a specific language

Execution of tasks and data movement between them assured by the **Runtime**

Runtimes

Advantages :

- Abstraction of architectures
- Portability of performance
- Good reactivity for load imbalance
- Natural look ahead

Challenge:

At the moment, runtimes are efficient for model of architectures

- DAGuE for large hierarchical architectures
- StarPU for heterogeneous architectures

Runtimes

Advantages :

- Abstraction of architectures
- Portability of performance
- Good reactivity for load imbalance
- Natural look ahead

Challenge:

At the moment, runtimes are efficient for model of architectures

- DAGuE for large hierarchical architectures
- StarPU for heterogeneous architectures

Summary

Context and Motivations
- $A = LU$
- $PA = LU$

LU Decomposition Without Pivoting ($A = LU$)

LU Decomposition with Partial Pivoting ($PA = LU$)
- Panel Factorization
- Update of Trailing Sub-matrix

Performance

Why LU decomposition

In order to solve square systems of linear equations:

$$Ax = b$$

Use LU decomposition (Gaussian elimination):

$$A = LU$$

Where L is a lower triangular matrix with the identity diagonal and U an upper triangular matrix.
Then solve:

$$Ly = b \text{ then } Ux = y$$

LU algorithm

Let be A a $n * n$ square matrix

For k from 1 to n

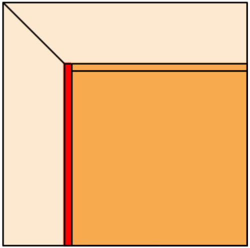
  For i from $k+1$ to n

    $a_{i,k} = a_{i,k}/a_{k,k}$

  For i from $k+1$ to n

    For j from $k+1$ to n

      $a_{i,j} = a_{i,j} - a_{i,k} *_{k,j}$



Summary

Context and Motivations

- $A = LU$
- $PA = LU$

LU Decomposition Without Pivoting ($A = LU$)

LU Decomposition with Partial Pivoting ($PA = LU$)

- Panel Factorization
- Update of Trailing Sub-matrix

Performance

LU decomposition problem

Let be A a $n * n$ square matrix
For k from 1 to n

For i from $k + 1$ to n
$a_{i,k} = a_{i,k}/a_{k,k}$
For i from $k + 1$ to n
For j from $k + 1$ to n
$a_{i,j} = a_{i,j} - a_{i,k} *_{k,j}$

- $a_{k,k}$ may be equal or close to zero
- Numerical value may be deteriorated due to fixed precision used by computers

$\Rightarrow$ LU decomposition is not stable

LU decomposition problem

Let be A a $n * n$ square matrix
For k from 1 to n

For i from $k+1$ to n
$a_{i,k} = a_{i,k}/a_{k,k}$
For i from $k+1$ to n
For j from $k+1$ to n
$a_{i,j} = a_{i,j} - a_{i,k} *_{k,j}$

- $a_{k,k}$ may be equal or close to zero
- Numerical value may be deteriorated due to fixed precision used by computers

$\Rightarrow$ LU decomposition is not stable

LU decomposition problem

Let be A a $n * n$ square matrix
For k from 1 to n
  Search for pivot then swap
  For i from $k+1$ to n
    $a_{i,k} = a_{i,k}/a_{k,k}$
  For i from $k+1$ to n
    For j from $k+1$ to n
      $a_{i,j} = a_{i,j} - a_{i,k} *_{k,j}$

- $a_{k,k}$ may be equal or close to zero
- Numerical value may be deteriorated due to fixed precision used by computers

$\Rightarrow$ LU decomposition is not stable

Solution is pivoting

Partial Pivoting Algorithm

The partial pivoting consist to look for the element with the maximal absolute value on the k^{th} column from $a_{k,k}$, then swap its row with the row consisting $a_{k,k}$.

The factorization amount to $PA = LU$

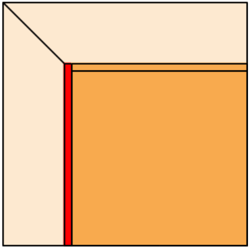
The partial pivoting is:

- Practically stable and accurate
- Commonly used in the scientific community
- Used in the LINPACK benchmark to rank the TOP 500 super-computers

LU implementation ($PA = LU$)

Software/Algorithms follow hardware evolution in time

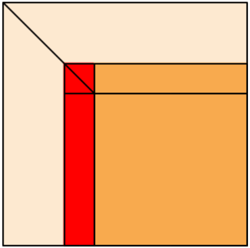
- 70's - LinPACK, vector operations
- 80's - LAPACK, block, cache-friendly
- 90's - ScaLAPACK, distributed memory



LU implementation ($PA = LU$)

Software/Algorithms follow hardware evolution in time

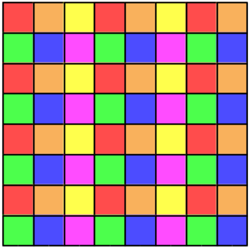
- 70's - LinPACK, vector operations
- 80's - LAPACK, block, cache-friendly
- 90's - ScaLAPACK, distributed memory



LU implementation ($PA = LU$)

Software/Algorithms follow hardware evolution in time

- 70's - LinPACK, vector operations
- 80's - LAPACK, block, cache-friendly
- 90's - ScaLAPACK, distributed memory



Summary

Context and Motivations

$A = LU$

$PA = LU$

LU Decomposition Without Pivoting ($A = LU$)

LU Decomposition with Partial Pivoting ($PA = LU$)

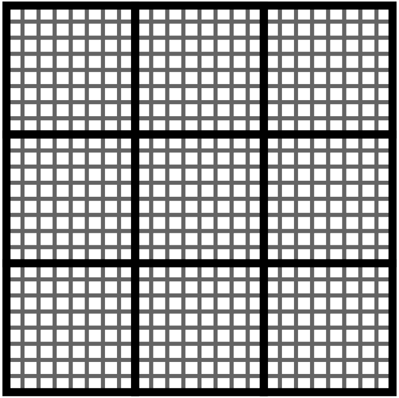
Panel Factorization

Update of Trailing Sub-matrix

Performance

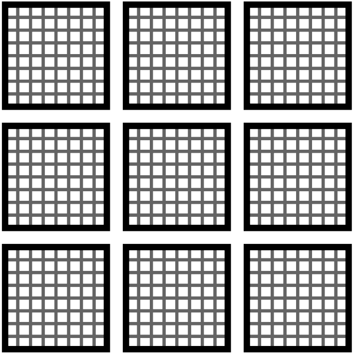
Task flow LU ($A = LU$)

In order to cope with the task flow model, linear algebra algorithm are expressed in terms of tasks operating on fine grain squares sub-matrices, also called tiles.



Task flow LU ($A = LU$)

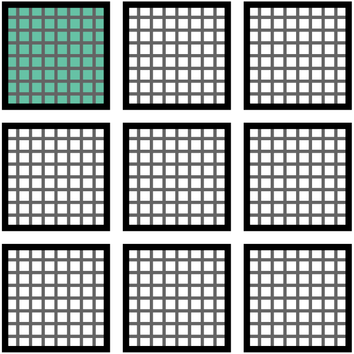
In order to cope with the task flow model, linear algebra algorithm are expressed in terms of tasks operating on fine grain squares sub-matrices, also called tiles.



Tiles are not contiguous in memory

Task flow LU ($A = LU$)

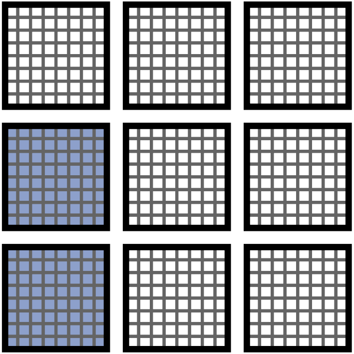
In order to cope with the task flow model, linear algebra algorithm are expressed in terms of tasks operating on fine grain squares sub-matrices, also called tiles.

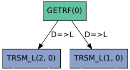


Task to factorize diagonal tiles: GETRF

Task flow LU ($A = LU$)

In order to cope with the task flow model, linear algebra algorithm are expressed in terms of tasks operating on fine grain squares sub-matrices, also called tiles.

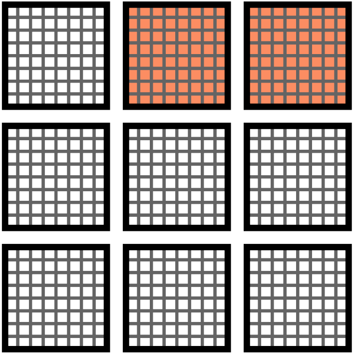


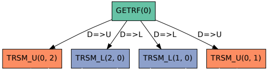


Task to update other panel tiles: TRSM_L

Task flow LU ($A = LU$)

In order to cope with the task flow model, linear algebra algorithm are expressed in terms of tasks operating on fine grain squares sub-matrices, also called tiles.

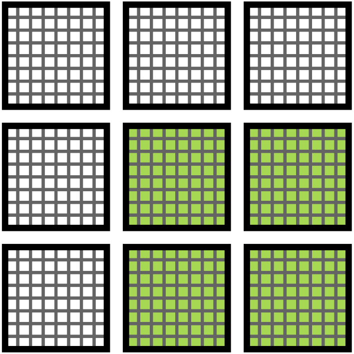


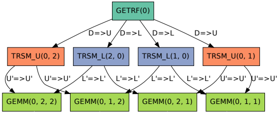


Task to update eliminated rows tiles: TRSM_U

Task flow LU ($A = LU$)

In order to cope with the task flow model, linear algebra algorithm are expressed in terms of tasks operating on fine grain squares sub-matrices, also called tiles.

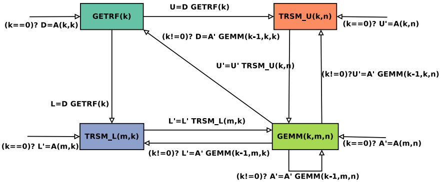


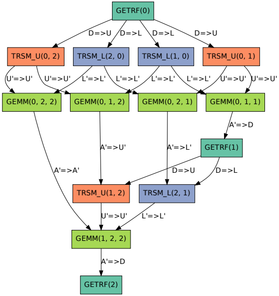


Task to update trailing sub-matrix: GEMM

Task flow LU ($A = LU$)

In order to cope with the task flow model, linear algebra algorithm are expressed in terms of tasks operating on fine grain squares sub-matrices, also called tiles.





Summary

Context and Motivations

- $A = LU$
- $PA = LU$

LU Decomposition Without Pivoting ($A = LU$)

LU Decomposition with Partial Pivoting ($PA = LU$)

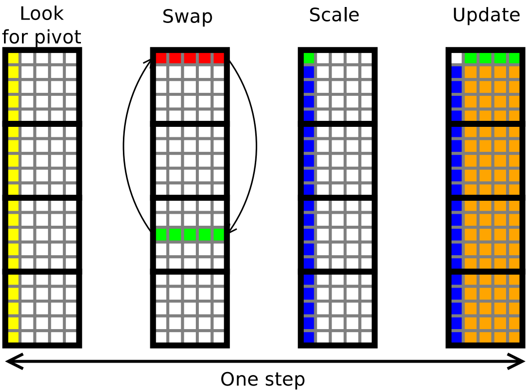
- Panel Factorization
- Update of Trailing Sub-matrix

Performance

Summary

Panel Factorization Problems

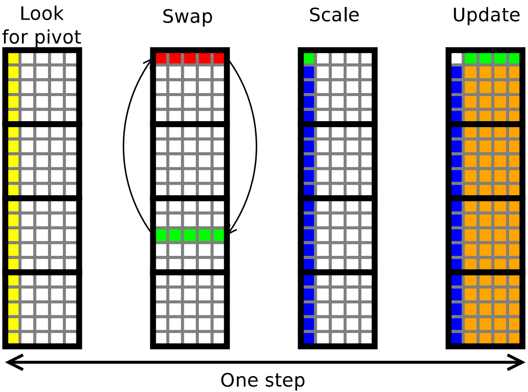
Tasks of panel factorization



- Look for a pivot is distributed over several tiles
- Tasks are fine grained

Panel Factorization Problems

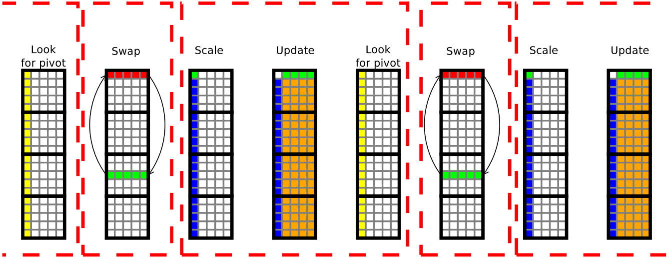
Tasks of panel factorization



- Look for a pivot is distributed over several tiles
- Tasks are fine grained

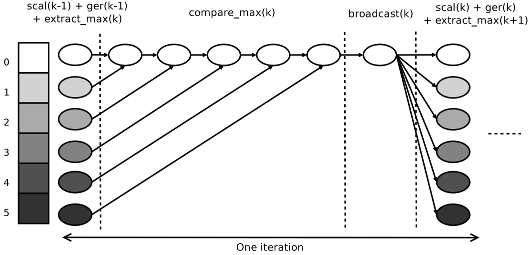
Panel Factorization Problems

Reduce fine grained tasks



Panel Factorization Problems

Natural task flow of panel factorization



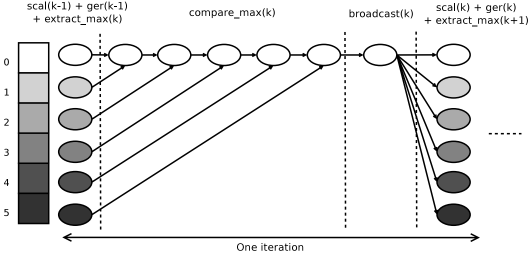
⇒ Serialized task flow!

Optimizations:

Use *all_reduce* operation (using Bruck's algorithm)

Panel Factorization Problems

Natural task flow of panel factorization



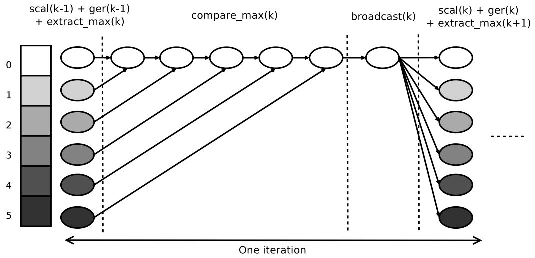
$\Rightarrow$ Serialized task flow!

Optimizations:

Use *all_reduce* operation (using Bruck's algorithm)

Panel Factorization Problems

Natural task flow of panel factorization



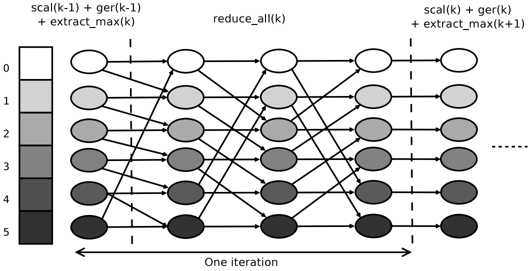
$\Rightarrow$ Serialized task flow!

Optimizations:

Use *all_reduce* operation (using Bruck's algorithm)

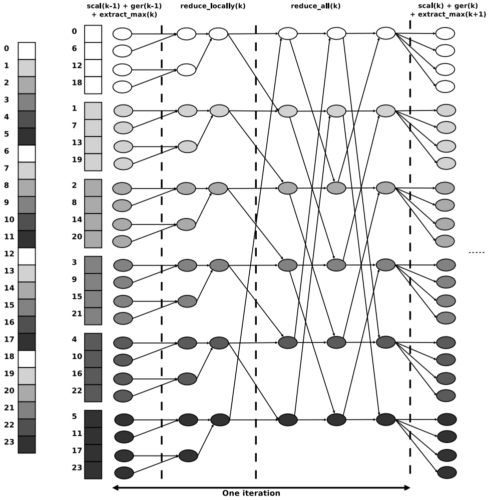
Panel Factorization Problems

Optimized task flow of panel factorization for distributed architectures



Panel Factorization Problems

Optimized task flow of panel factorization for hierarchical architectures



Summary

Context and Motivations

$A = LU$

$PA = LU$

LU Decomposition Without Pivoting ($A = LU$)

LU Decomposition with Partial Pivoting ($PA = LU$)

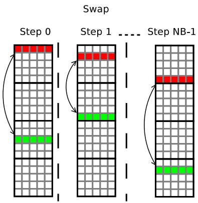
Panel Factorization

Update of Trailing Sub-matrix

Performance

Update Problems

Update trailing sub-matrix



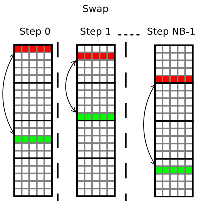
The upper tile exchange rows with other tile depending on pivots values.

Problem

- Dynamic decision for a static DAG $\rightarrow$ Generate tasks for all possible communications?
- Pivots implies swaps in a specific order $\rightarrow$ Use permutation instead of pivots
- Serialized communications $\rightarrow$ Separate Swap from/into upper tile

Update Problems

Update trailing sub-matrix



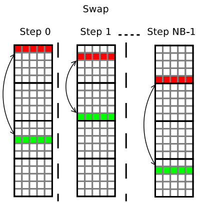
The upper tile exchange rows with other tile depending on pivots values.

Problem

- Dynamic decision for a static DAG → Generate tasks for all possible communications?
- Pivots implies swaps in a specific order → Use permutation instead of pivots
- Serialized communications → Separate Swap from/into upper tile

Update Problems

Update trailing sub-matrix



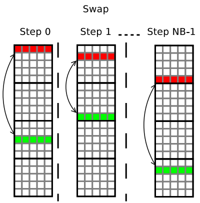
The upper tile exchange rows with other tile depending on pivots values.

Problem

- Dynamic decision for a static DAG → Generate tasks for all possible communications?
- Pivots implies swaps in a specific order → Use permutation instead of pivots
- Serialized communications → Separate Swap from/into upper tile

Update Problems

Update trailing sub-matrix



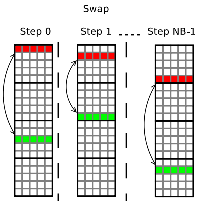
The upper tile exchange rows with other tile depending on pivots values.

Problem

- Dynamic decision for a static DAG → Generate tasks for all possible communications?
- Pivots implies swaps in a specific order → Use permutation instead of pivots
- Serialized communications → Separate Swap from/into upper tile

Update Problems

Update trailing sub-matrix



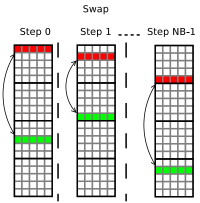
The upper tile exchange rows with other tile depending on pivots values.

Problem

- Dynamic decision for a static DAG $\rightarrow$ Generate tasks for all possible communications?
- Pivots implies swaps in a specific order $\rightarrow$ Use permutation instead of pivots
- Serialized communications $\rightarrow$ Separate Swap from/into upper tile

Update Problems

Update trailing sub-matrix



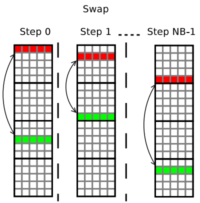
The upper tile exchange rows with other tile depending on pivots values.

Problem

- Dynamic decision for a static DAG $\rightarrow$ Generate tasks for all possible communications?
- Pivots implies swaps in a specific order $\rightarrow$ Use permutation instead of pivots
- Serialized communications $\rightarrow$ Separate Swap from/into upper tile

Update Problems

Update trailing sub-matrix



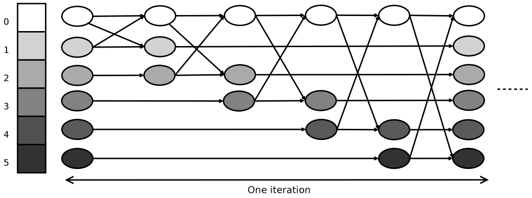
The upper tile exchange rows with other tile depending on pivots values.

Problem

- Dynamic decision for a static DAG $\rightarrow$ Generate tasks for all possible communications?
- Pivots implies swaps in a specific order $\rightarrow$ Use permutation instead of pivots
- Serialized communications $\rightarrow$ Separate Swap from/into upper tile

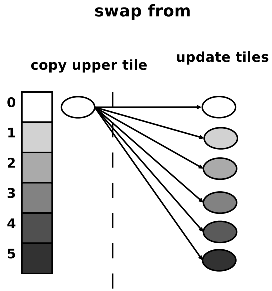
Update Problems

Natural task flow of one swap in update



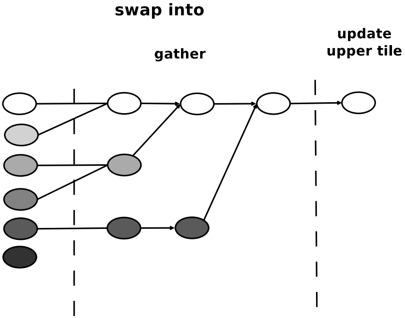
Update Problems

Optimized task flow of swaps of update for distributed architectures



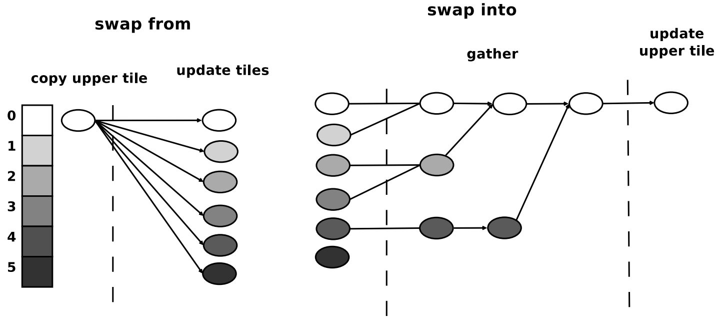
Update Problems

Optimized task flow of swaps of update for distributed architectures



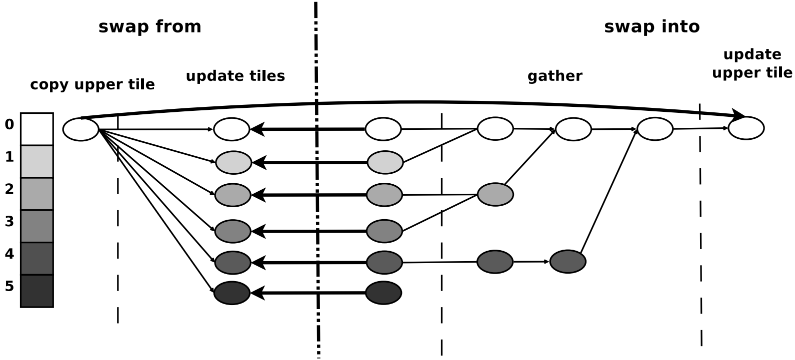
Update Problems

Optimized task flow of swaps of update for distributed architectures



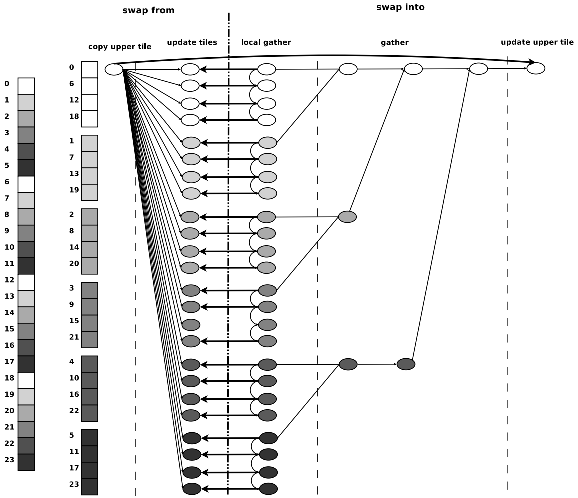
Update Problems

Optimized task flow of swaps of update for distributed architectures



Update Problems

Optimized task flow of swaps of update for hierarchical architectures



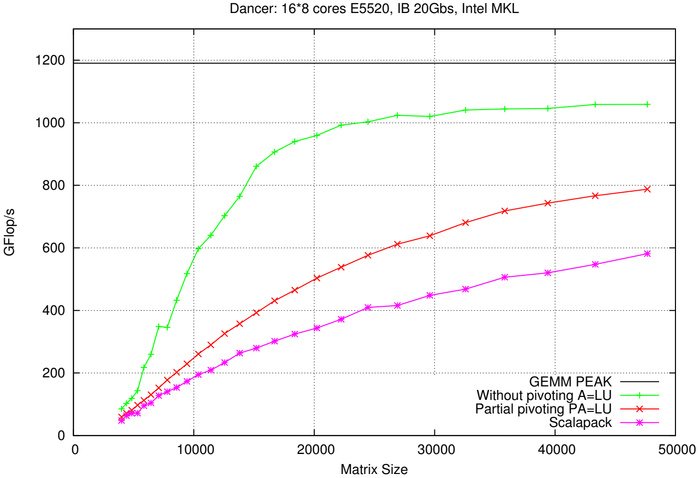
Summary

Context and Motivations
- $A = LU$
- $PA = LU$

LU Decomposition Without Pivoting ($A = LU$)

LU Decomposition with Partial Pivoting ($PA = LU$)
- Panel Factorization
- Update of Trailing Sub-matrix

Performance


Dancer: 16*8 cores E5520, IB 20Gbs, Intel MKL
GFlop/s
Matrix Size
0
200
400
600
800
1000
1200
0
10000
20000
30000
40000
50000
GEMM PEAK
Without pivoting A=LU
Partial pivoting PA=LU
Scalapack

Conclusion and future work

Conclusion :

- Implemented static pivoting ($A = LU$) on DAGuE and StarPU
- Implemented partial pivoting ($PA = LU$) on DAGuE and StarPU
- Exhibited the feasibility of partial pivoting on task flow model
- Obtained encouraging performances with partial pivoting over DAGuE
- Performances of partial pivoting over StarPU still not exploitable

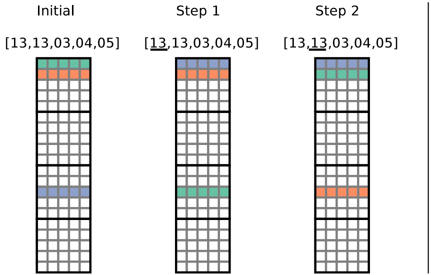
Future work :

- Integrate GPU pivoting operations to DAGuE and StarPU
- Try other startegies of panel factorizations on several methods
- Build a new benchmark based on the DAGuE partial pivoting

Thank you !

ANNEXE

Using permutations instead pivots



ANNEXE

Using permutations instead pivots