Adaptive Parallel Exact dense LU factorization

Ziad SULTAN

15 mai 2013

JNCF2013, Université de Grenoble

Outline

Outline

Gaussian elimination in **Dense** Computer algebra

Dense

- benchmarking of supercomputers (www.top500.org)
- basis of linear algebra

Sparse

Large sparse matrix problems $\rightarrow$ smaller dense problems (still large !)

- Sparse Iterative :
  Induce dense elimination on blocs of iterated vectors
  (Krylov, Lanczos, smith normal form)
- Sparse Direct :
  Switch to dense after fill-in [FGB]

Outline

Gaussian elimination in numerical computation

pivoting strategies

- search for best pivot
  $\rightarrow$ good numerical stability
- good data locality
- Reduce the fill-in
  $\rightarrow$ reduce additional memory needs
  $\rightarrow$ reduce induced computation costs

Exact gaussian elimination applications

Exact

- Rank
  → Algebraic topology (smith normal form)
- Rank Profile
  → Grobner basis computation [FGB]
  → Computational number theory [Stein]
- Characteristic Polynomial
  → Graph Theory [G. Royle]
- Coding theory
  → Semi-fields

Rank profile

Row/Column rank profile

Definition

lexico-graphically smallest sequence of r row/column indices s.t. the corresponding rows/columns of A are linearly independant.

Generic rank profile

If its first r leading principal minors are non zero

example :

the sequence $\{1,...,r\}$ is the row rank profile of a generic rank profile matrix

Outline

Optimized building block in Dense linear algebra

Matrix multiplication

Algorithmic complexity : recursive

$\rightarrow$ Strassen $O(n^{2.8}), ..., O(n^{\omega})$

Optimized hardware implementation : iterative

$\rightarrow$ pipeline, SSE, AVX, ...

Implementation : block versions cascading

- cache optimization
- reduce dependencies on the bus speed
  - faster computation for blocks loaded in cache

Gaussian elimination concerns

Same concerns as M.M. = block versions

Implementation optimization
$\rightarrow$ benefits from matrix multiplication
Reduce dependencies on bus speed
(cache optimization)

Possible best versions adapted for parallel computing

Tiled iterative implementation
block recursive implementation

Exact gaussian elimination adapted for Parallel computing

block versions trade-off

Common point less memory accesses
if block size fits the cache
$\rightarrow N^3/B$ memory accesses.
(N dimension of the matrix, B the block size)

Trade-off
block recursive :
$\rightarrow$ More adaptative
tiled iterative :
$\rightarrow$ less synchronizations
$\rightarrow$ Historically, It's more difficult to parallelize recursive implementation with existing model of Parallel computing (OpenMP, ...)

State of the art

State of the art

Sequential Exact :

$\rightarrow$ FFLAS-FFPACK, M4RI, within FGB

Parallel numeric :

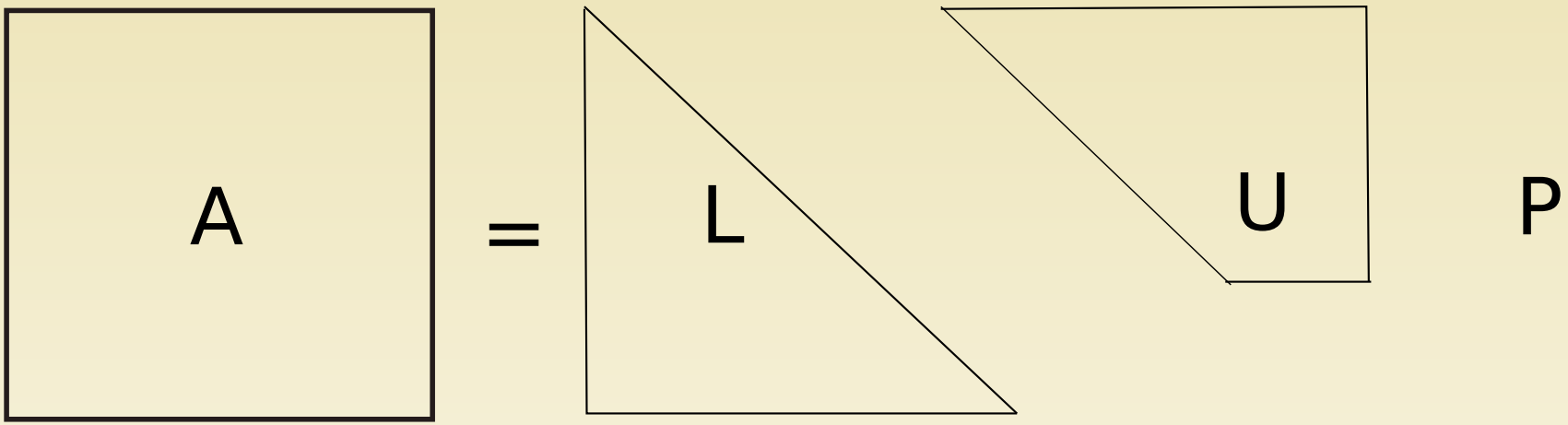
$\rightarrow$ ScaLAPACK, Plasma-Quark

Parallel exact : ? ?

$\rightarrow$ this work

Outline

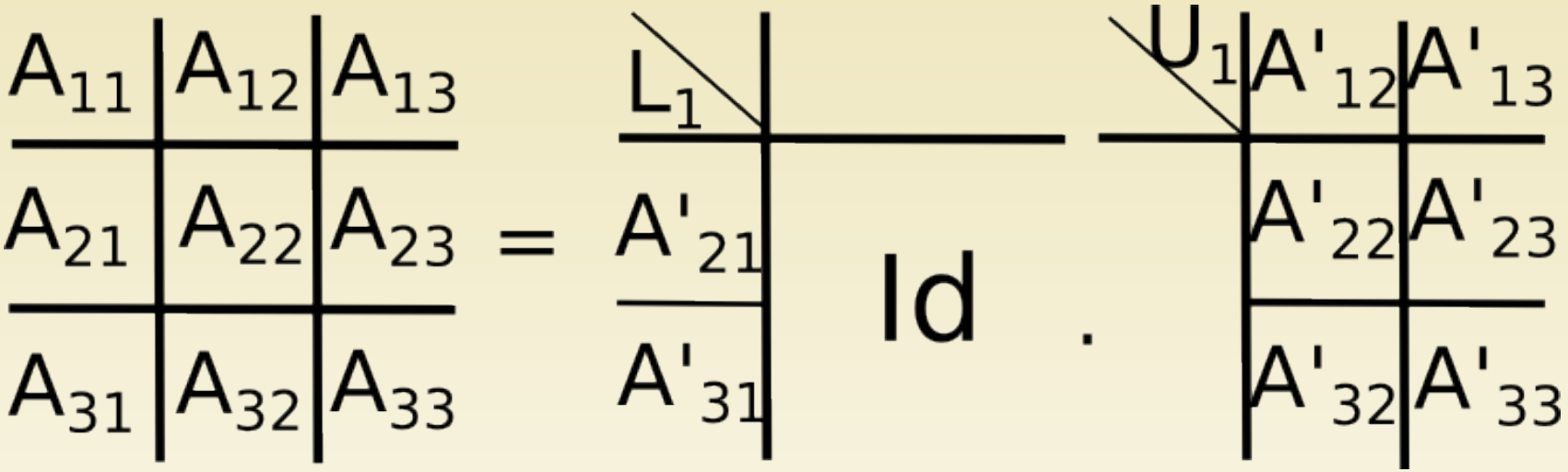
LU factorization of generic rank profile matrices



LU Decomposition applications

Solving System : $A.x = b; L.(U.x) = b; L.(y) = b$
Rank : Rank(A) is the number of rows of U
Invert of A : $A^{-1} = U^{-1}.L^{-1}$
Determinant : $det(A) = \pm det(U)$
row or column Rank Profile : given by positions of row or column permutations

Tiled iterative LU decomposition

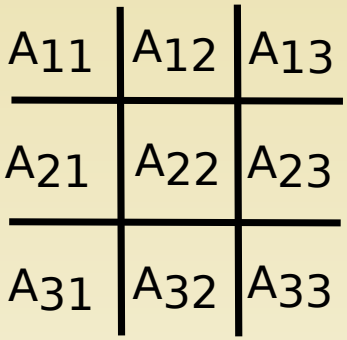


LU decomposition on first block $A_{11} = L_1.U_1$

updates : $\boxed{A'_{21} = A_{21}.U_1^{-1}}$; $\boxed{A'_{31} = A_{31}.U_1^{-1}}$

$\boxed{A'_{12} = L_1^{-1}.A_{12}}$; $\boxed{A'_{13} = L_1^{-1}.A_{13}}$; $\boxed{A'_{22} = A_{22} - A'_{21}.A'_{12}}$ ...

Tiled iterative LU Decomposition



Routines of FFLAS-FFPACK LIBRARY on a Finite Field Z/pZ

FTRSM (update blocks on same column and same row)

$$A_{ik} = A_{ik}.U_{kk}^{-1}$$
$$A_{ki} = L_{kk}^{-1}.A_{ki}$$

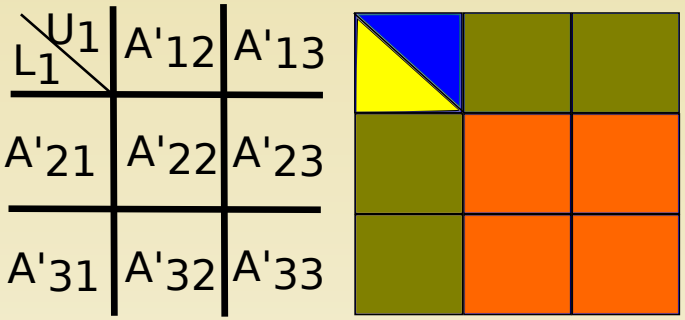
FGEMM (MM, update remaining blocks)

$$A_{ij} = A_{ij} - A_{ik}.A_{kj}$$

applyP (applying permutation matrix)

$$A_{ik} = A_{ik}.P_{kk}^{-1}$$

Tiled iterative LU Decomposition



Routines of FFLAS-FFPACK LIBRARY on a Finite Field Z/pZ

FTRSM (update blocks on same column and same row)

$$A_{ik} = A_{ik}.U_{kk}^{-1}$$
$$A_{ki} = L_{kk}^{-1}.A_{ki}$$

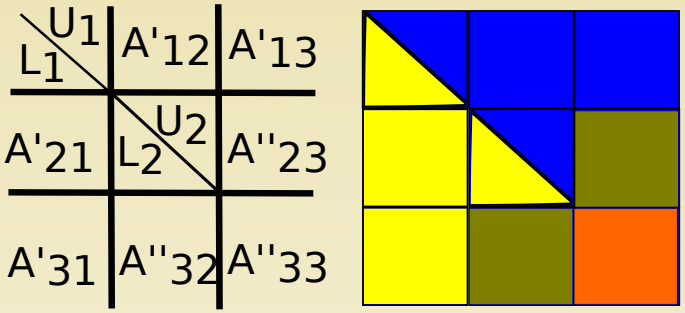
FGEMM (MM, update remaining blocks)

$$A_{ij} = A_{ij} - A_{ik}.A_{kj}$$

applyP (applying permutation matrix)

$$A_{ik} = A_{ik}.P_{kk}^{-1}$$

Tiled iterative LU Decomposition



Routines of FFLAS-FFPACK LIBRARY on a Finite Field Z/pZ

FTRSM (update blocks on same column and same row)

$$A_{ik} = A_{ik}.U_{kk}^{-1}$$
$$A_{ki} = L_{kk}^{-1}.A_{ki}$$

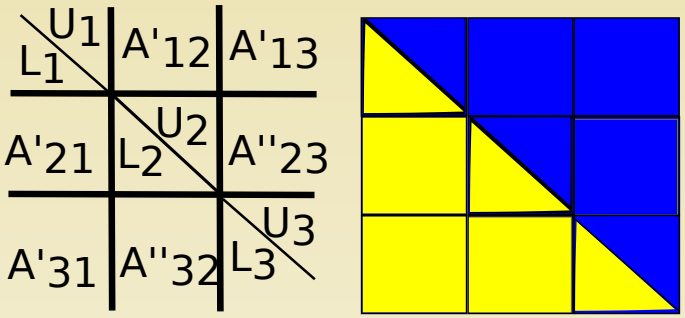
FGEMM (MM, update remaining blocks)

$$A_{ij} = A_{ij} - A_{ik}.A_{kj}$$

applyP (applying permutation matrix)

$$A_{ik} = A_{ik}.P_{kk}^{-1}$$

Tiled iterative LU Decomposition



Routines of FFLAS-FFPACK LIBRARY on a Finite Field Z/pZ

FTRSM (update blocks on same column and same row)

$$A_{ik} = A_{ik}.U_{kk}^{-1}$$
$$A_{ki} = L_{kk}^{-1}.A_{ki}$$

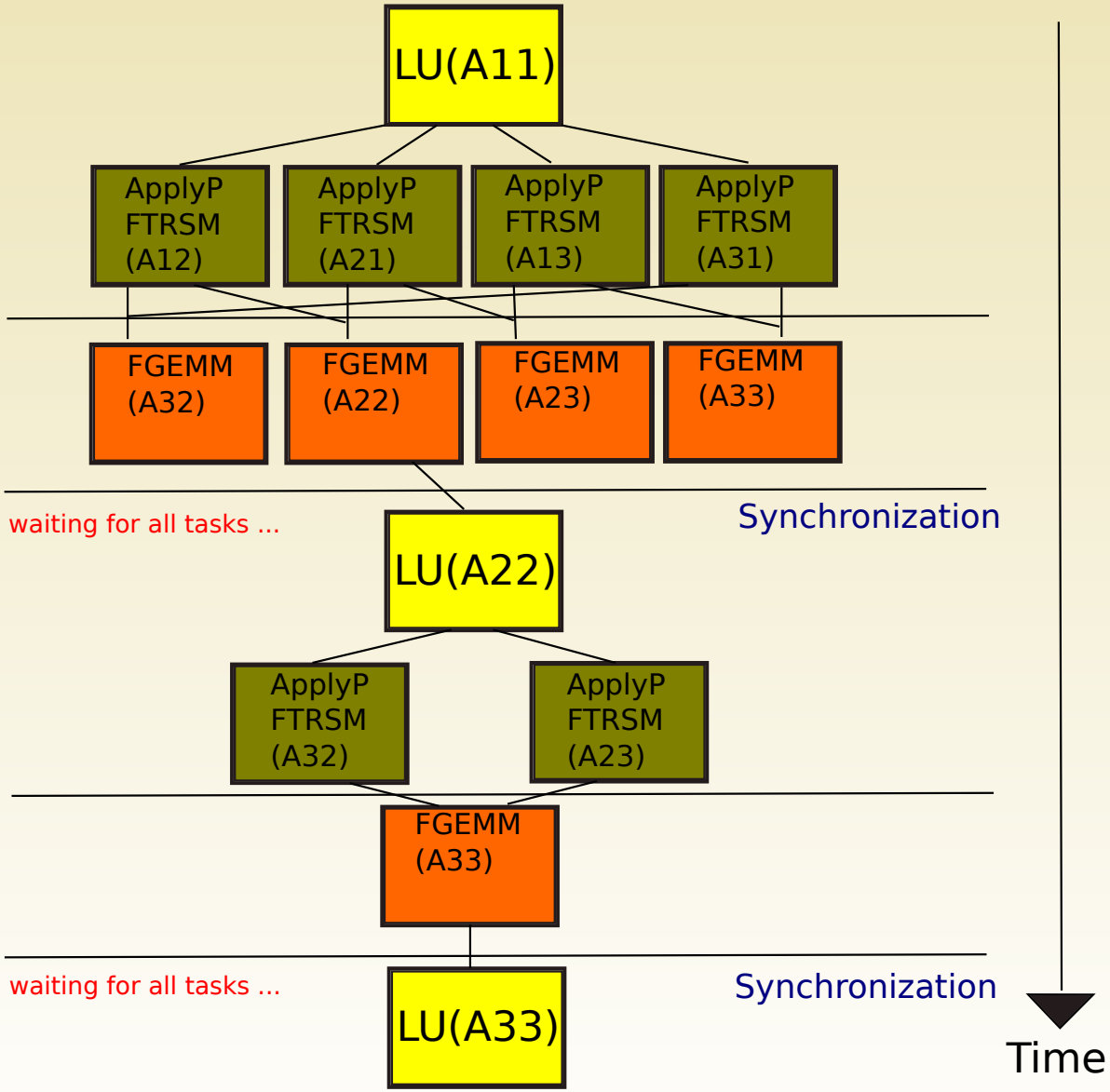
FGEMM (MM, update remaining blocks)

$$A_{ij} = A_{ij} - A_{ik}.A_{kj}$$

applyP (applying permutation matrix)

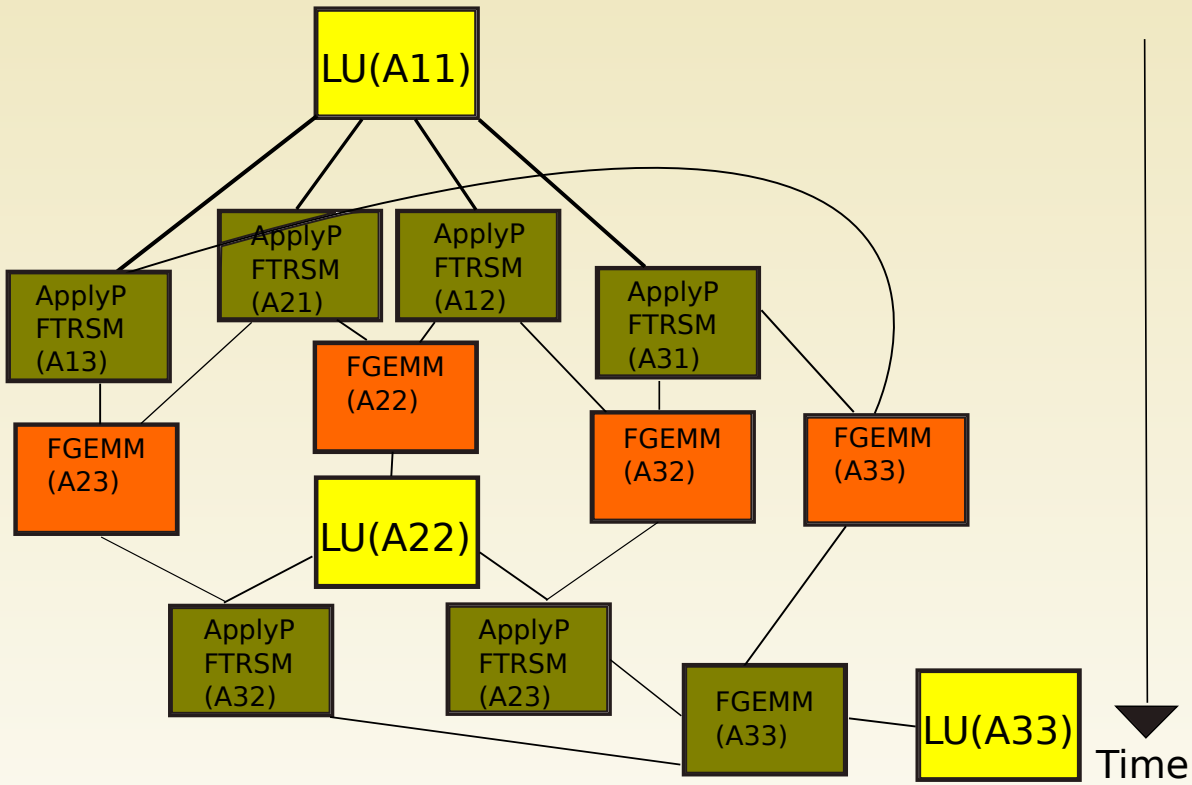
$$A_{ik} = A_{ik}.P_{kk}^{-1}$$

OpenMP parallel loop Synchronizations



```
for(k=0 ; k<nblocks ; k++){
    R = FFPACK : :LUdivine(...) ;
    #pragma omp parallel for shared(A, P)
    {
      #pragma omp for nowait
      for(i=k+1 ; i<nblocks ; i++)
        FFLAS : :ftrsm(...) ;
    }
    #pragma omp parallel for shared(A, P)
    for(i=k+1 ; i<nblocks ; i++){
      FFPACK : :applyP(...) ;
      FFLAS : :ftrsm(...) ;
    }
    #pragma omp parallel for shared(A, P, T)
    for(i=k+1 ; i<nblocks ; i++){
      #pragma omp parallel for shared(A )
      for(j=k+1 ; j<nblocks ; j++){
        FFLAS : :fgemm(...) ;}}
  }
```

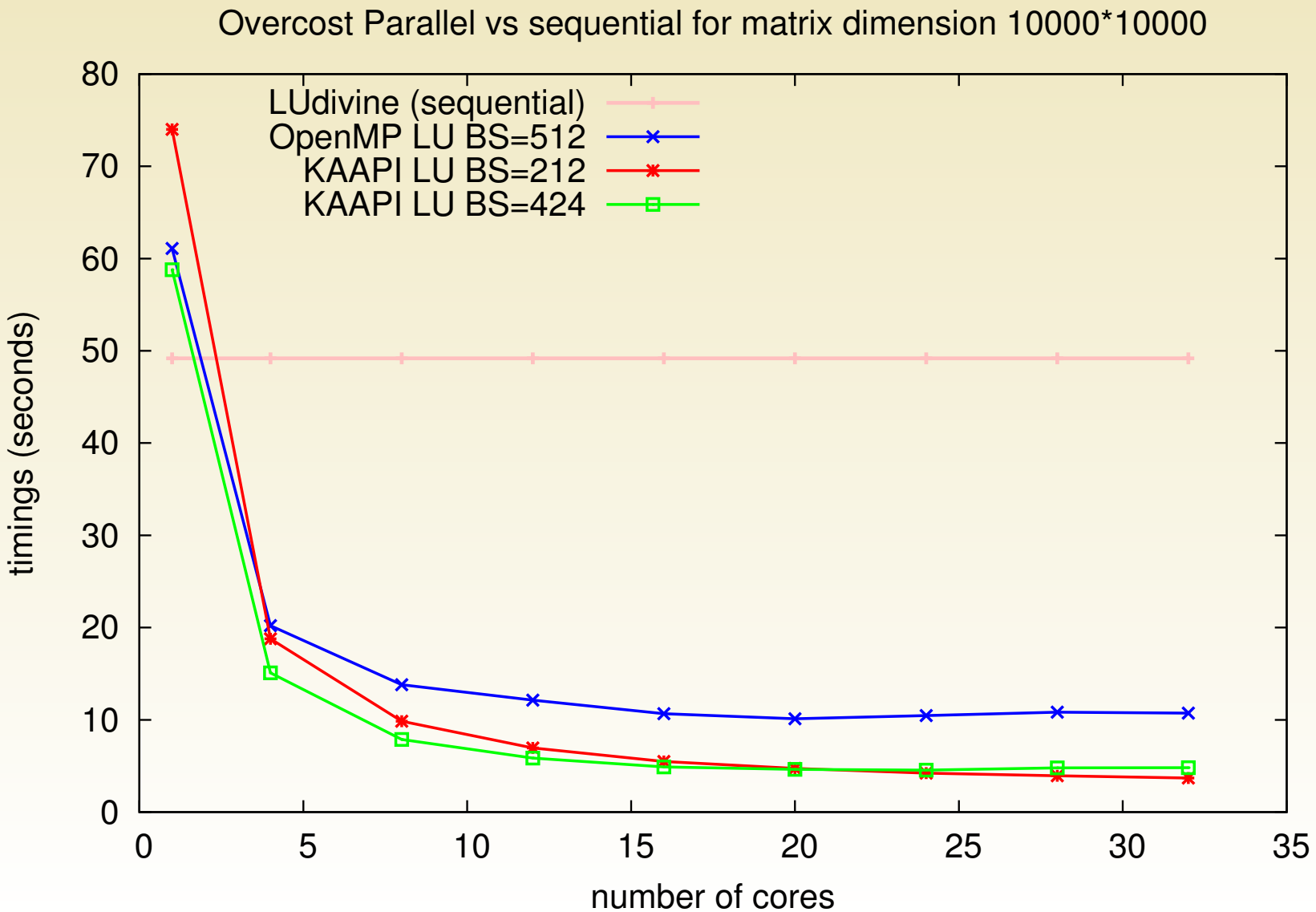
KAAPI dataflow scheduling for Tiled LUP



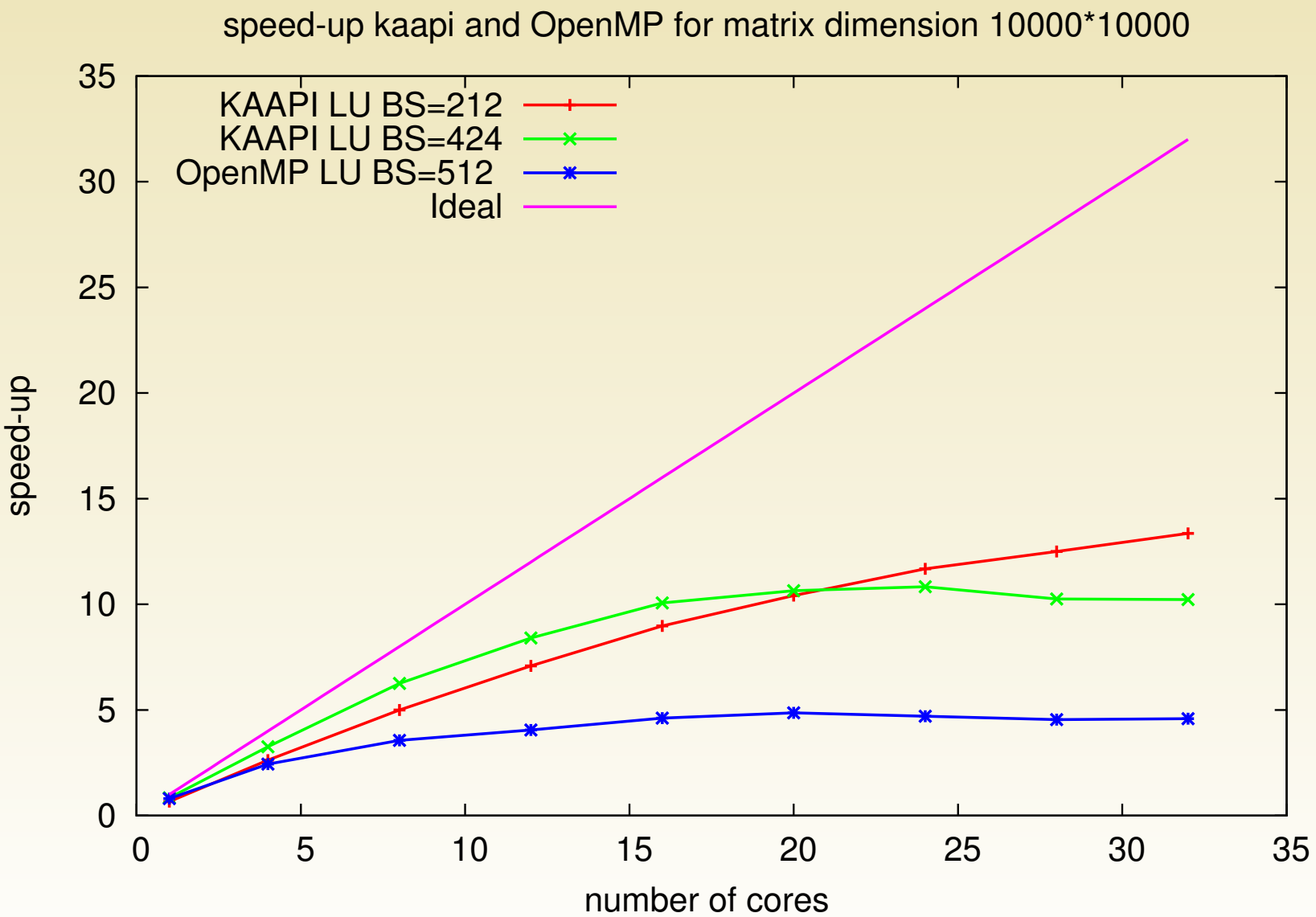
```
for(int k=0 ; k<nblocks ; k++){
    #pragma kaapi task readwrite(&A) write(&P, &Q)
    R = FFPACK : :LUdivine(...) ;
    for(int i=k+1 ; i<nblocks ; i++){
      #pragma kaapi task readwrite(&A) read(&A)
      FFLAS : :ftrsm(...) ;
    }
    for(int i=k+1 ; i<nblocks ; i++){
      #pragma kaapi task readwrite(&A) read(&P)
      FFPACK : :applyP(...) ;
      #pragma kaapi task readwrite(&A) read(&A)
      FFLAS : :ftrsm(...) ;
    }
    for(int i=k+1 ; i<nblocks ; i++){
       for(int j=k+1 ; j<nblocks ; j++){
       #pragma kaapi task readwrite(&A) read(&A)
      FFPACK : :fgemm(...) ; } }
}
```

KAAPI vs OpenMP

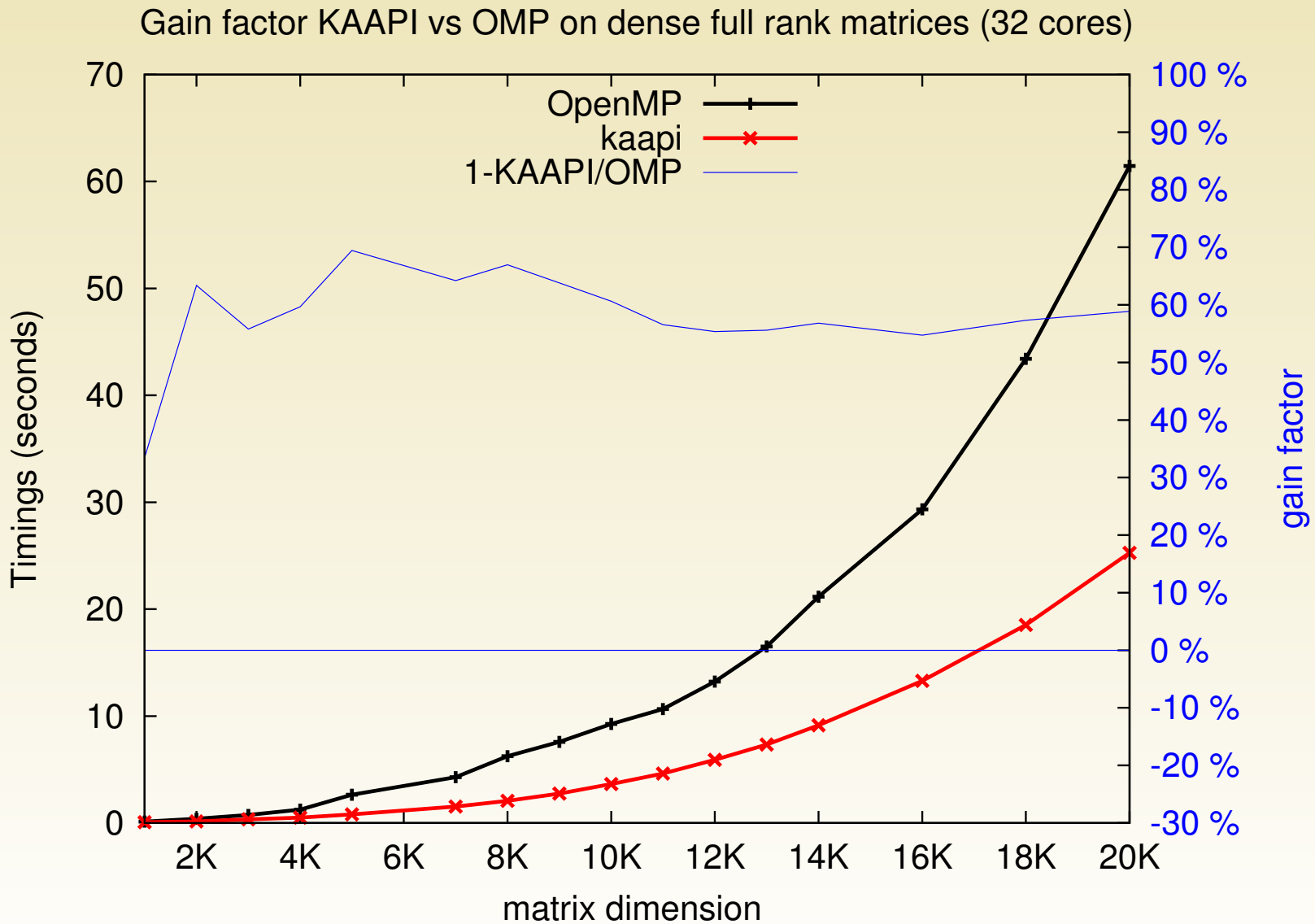
HPAC : Intel SandyBridge E5-4620 2.2Ghz, 32 cores, L3 cache(16384 KB). $(\mathbb{Z}/1009\mathbb{Z})$



KAAPI version speed-up

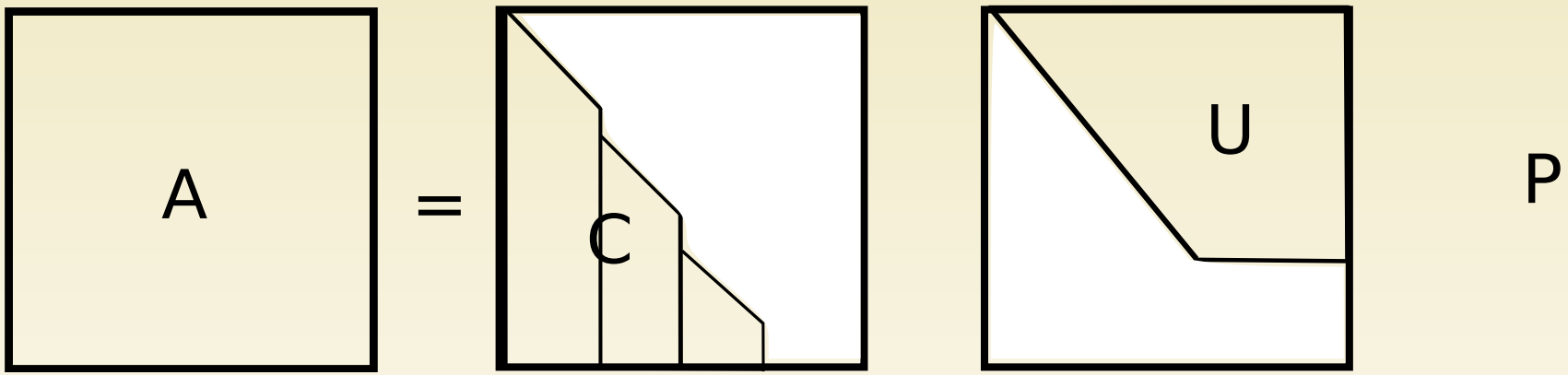


Parallelization overcost on LU algorithm

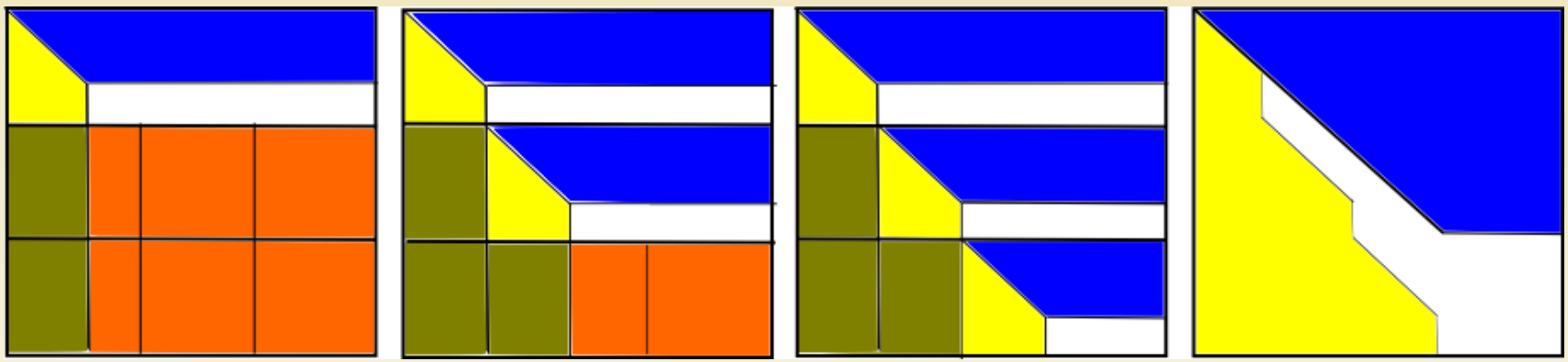


Outline

CUP decomposition (Rank deficient matrices)



block CUP decomposition

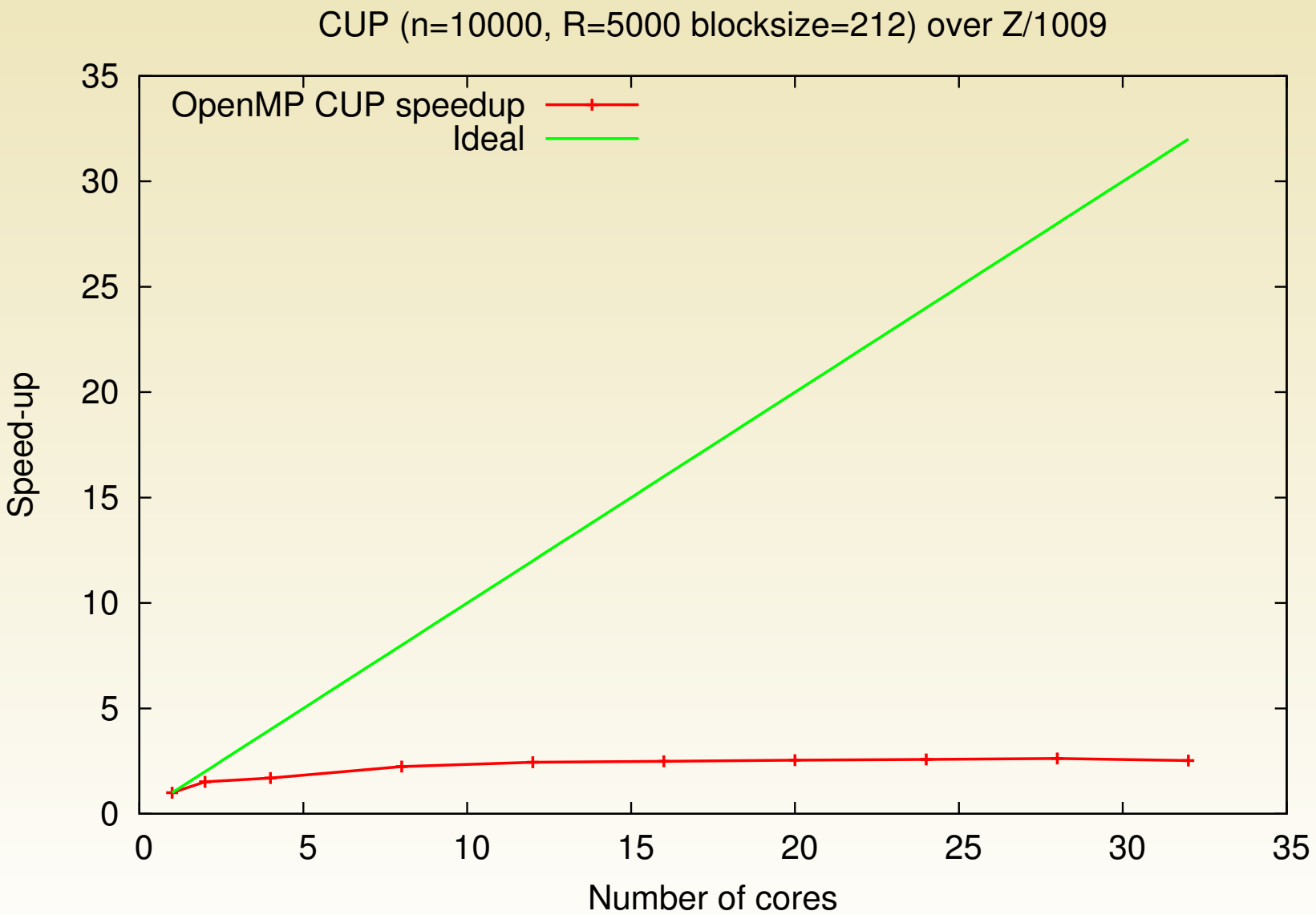


block CUP

Less parallelism

- some independent tasks removed
- big sequential costly task

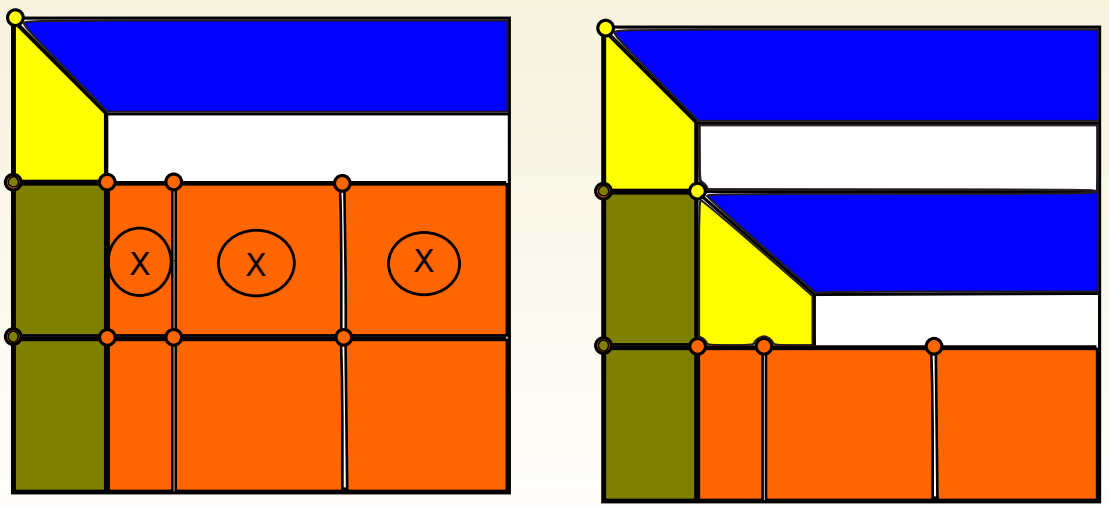
Parallelization of block CUP with OpenMP



Parallelization of block CUP with KAAPI Dynamic scheduling

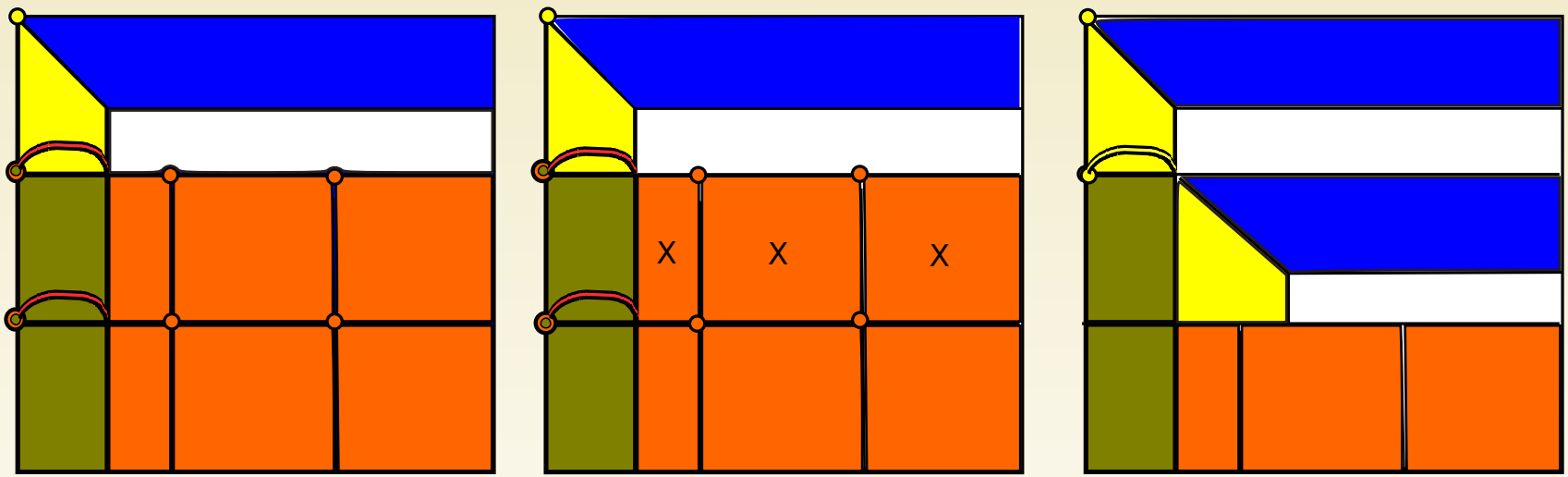
dependencies

- The graph of task dependency is calculated during runtime
- Dependency between tasks is done according to the referent of each task.
- In this implementation, the referent is the pointer of the block i.e. it's the pointer on the upper-left side of each block.



Parallelization of block CUP with KAAPI Static scheduling

The graph of task dependancy is precalculated before execution. (faster)

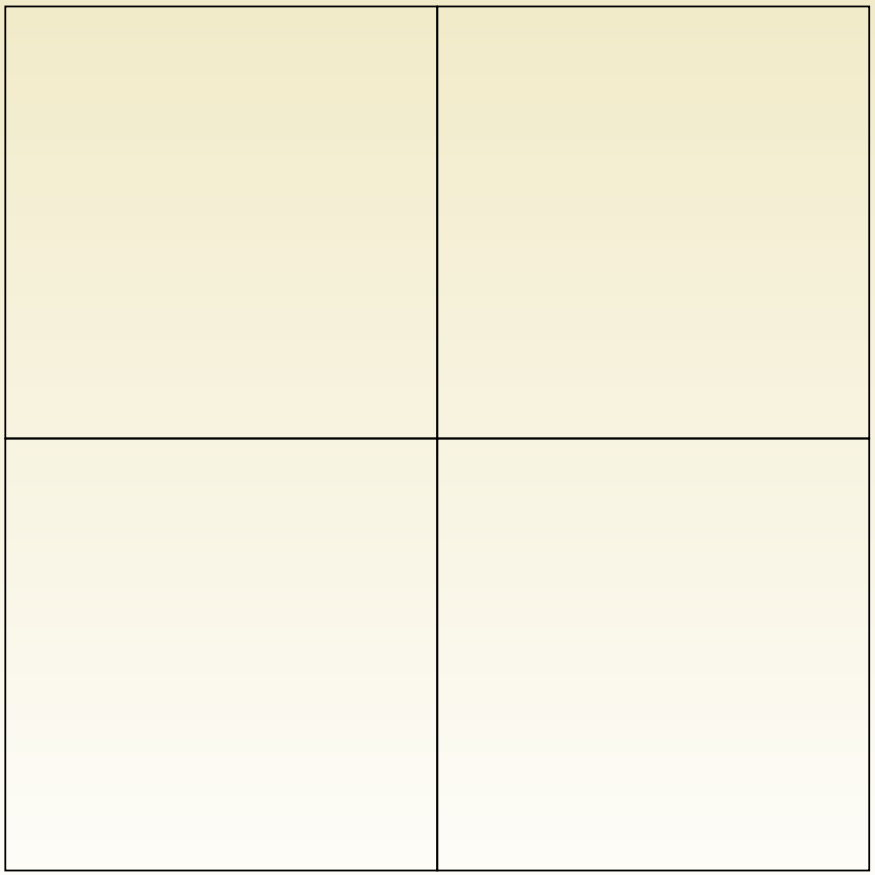


X is a task parameter, set as CW. CW mode for static scheduling is not defined yet in actual KAAPI version.

Outline

Algorithme PLUQ Quad-recursif

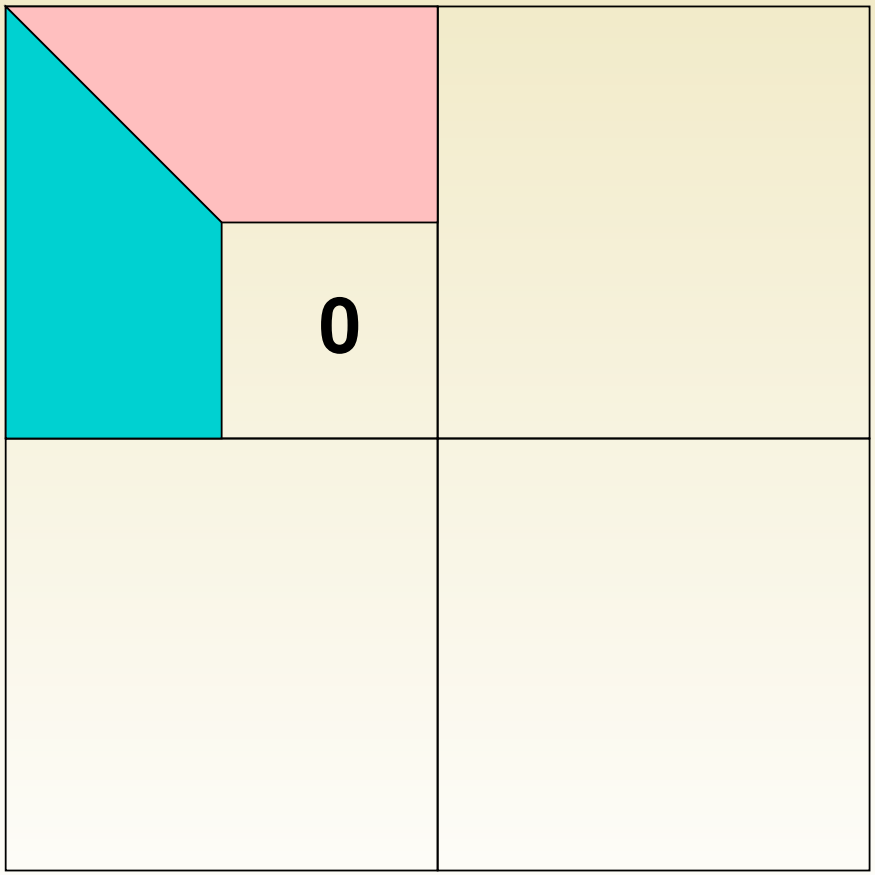
- Recursive cutting according to row and columns
- We can permut the blocks in a way that we can obtain row rank profile and column rank profile at the same time.
- rank profile of all leading sub-matrices



[Dumas, Pernet, Sultan, ISSAC 2013]

Algorithme PLUQ Quad-recursif

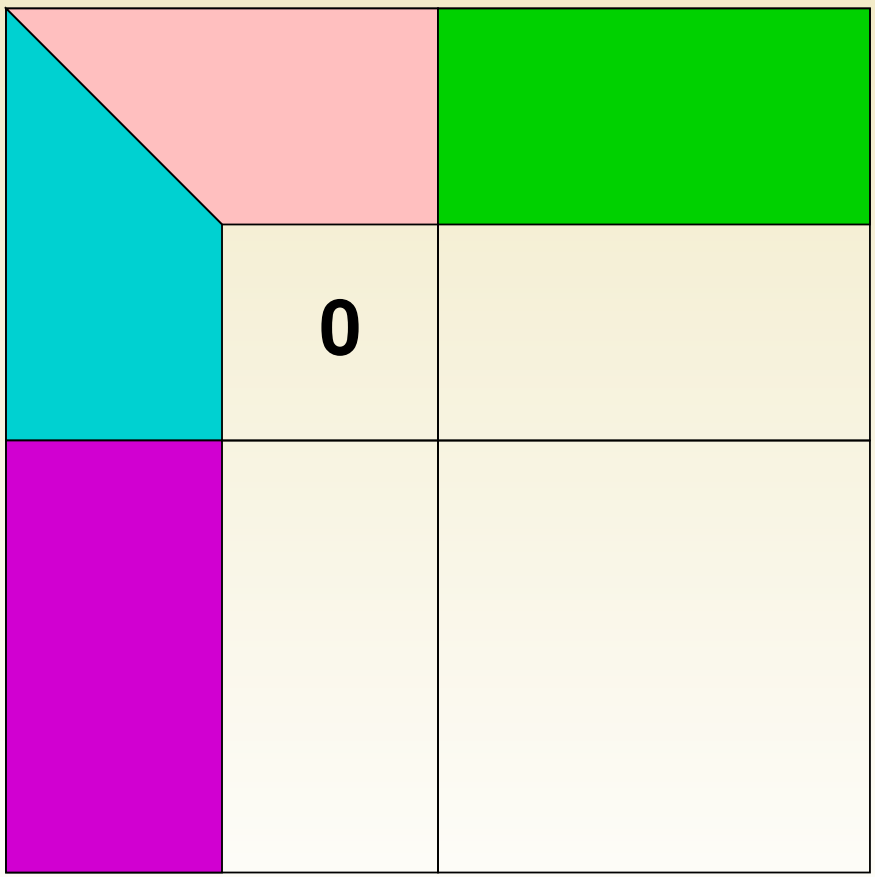
- Recursive cutting according to row and columns
- We can permut the blocks in a way that we can obtain row rank profile and column rank profile at the same time.
- rank profile of all leading sub-matrices



[Dumas, Pernet, Sultan, ISSAC 2013]

Algorithme PLUQ Quad-recursif

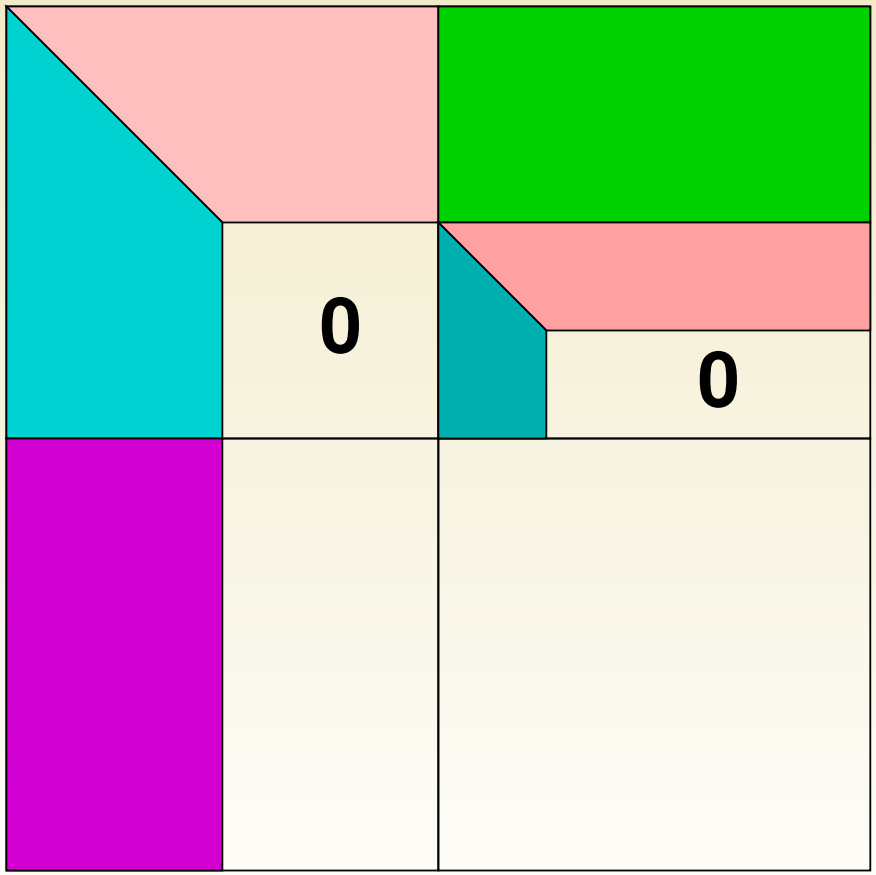
- Recursive cutting according to row and columns
- We can permut the blocks in a way that we can obtain row rank profile and column rank profile at the same time.
- rank profile of all leading sub-matrices



[Dumas, Pernet, Sultan, ISSAC 2013]

Algorithme PLUQ Quad-recursif

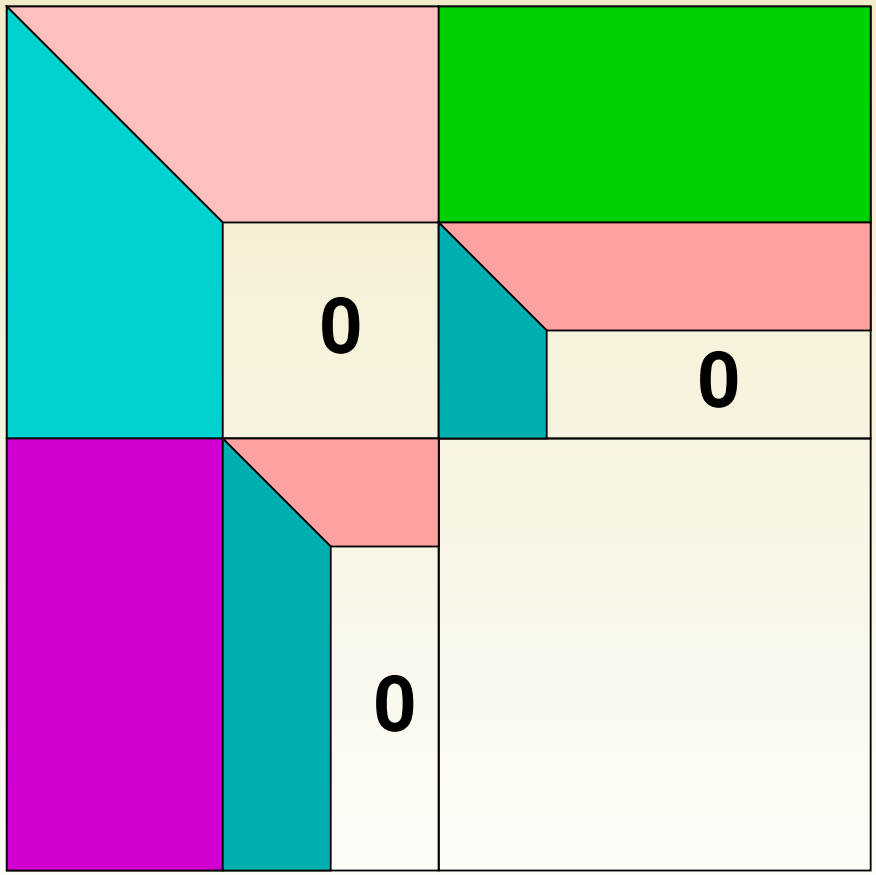
- Recursive cutting according to row and columns
- We can permut the blocks in a way that we can obtain row rank profile and column rank profile at the same time.
- rank profile of all leading sub-matrices



[Dumas, Pernet, Sultan, ISSAC 2013]

Algorithme PLUQ Quad-recursif

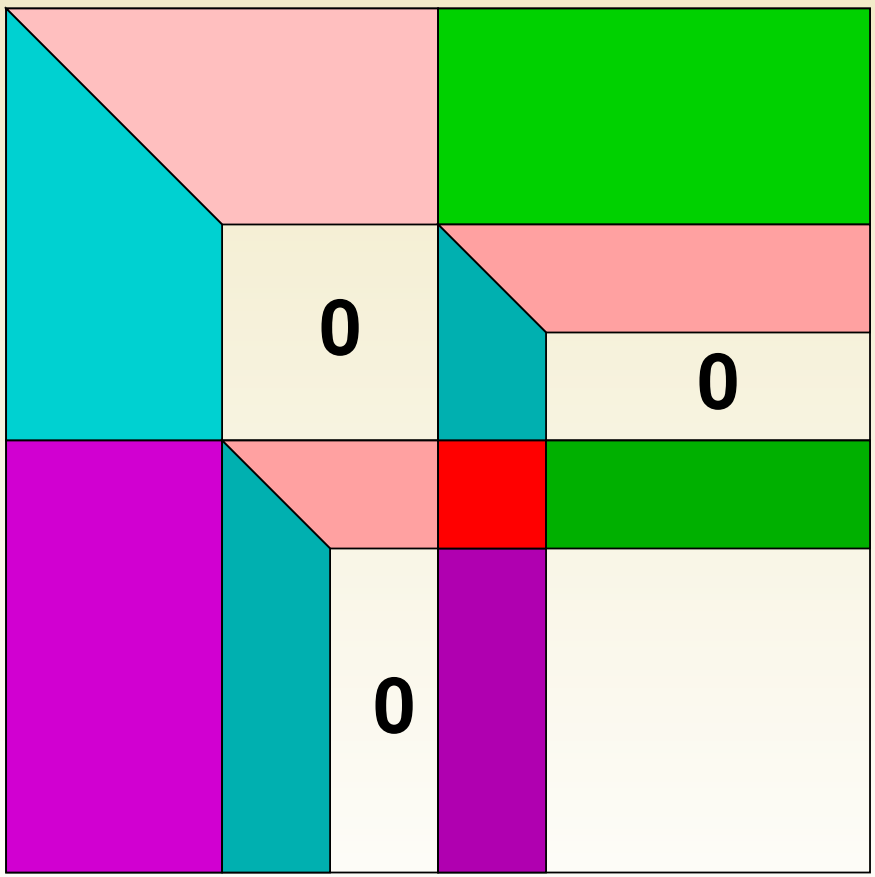
- Recursive cutting according to row and columns
- We can permut the blocks in a way that we can obtain row rank profile and column rank profile at the same time.
- rank profile of all leading sub-matrices



[Dumas, Pernet, Sultan, ISSAC 2013]

Algorithme PLUQ Quad-recursif

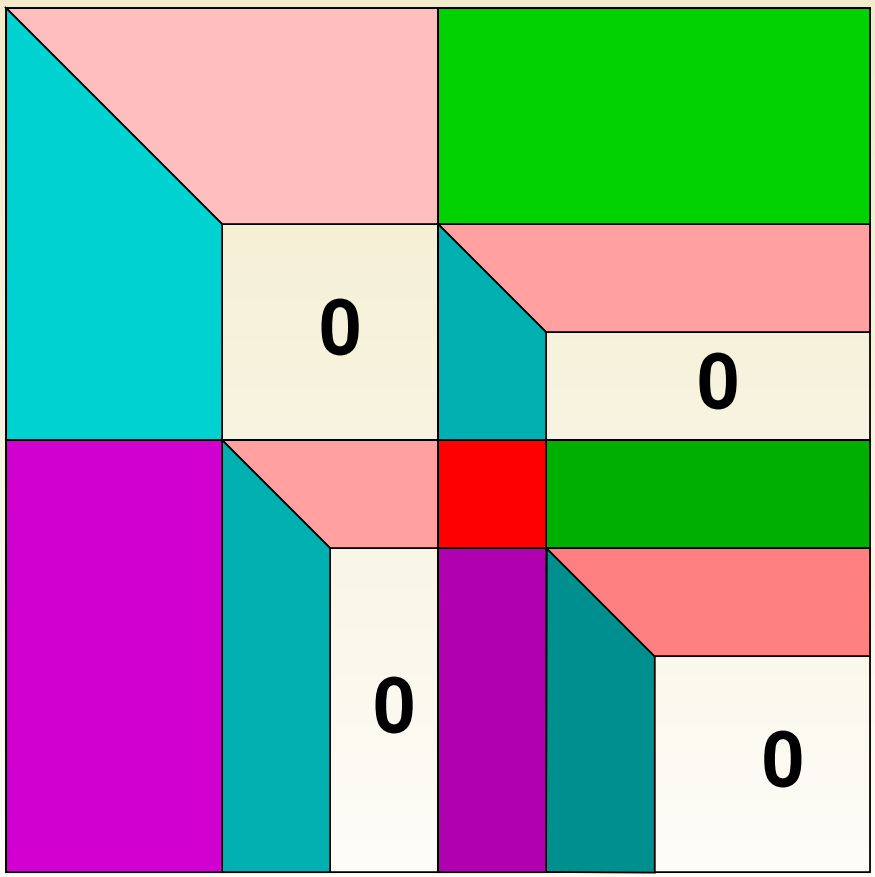
- Recursive cutting according to row and columns
- We can permut the blocks in a way that we can obtain row rank profile and column rank profile at the same time.
- rank profile of all leading sub-matrices



[Dumas, Pernet, Sultan, ISSAC 2013]

Algorithme PLUQ Quad-recursif

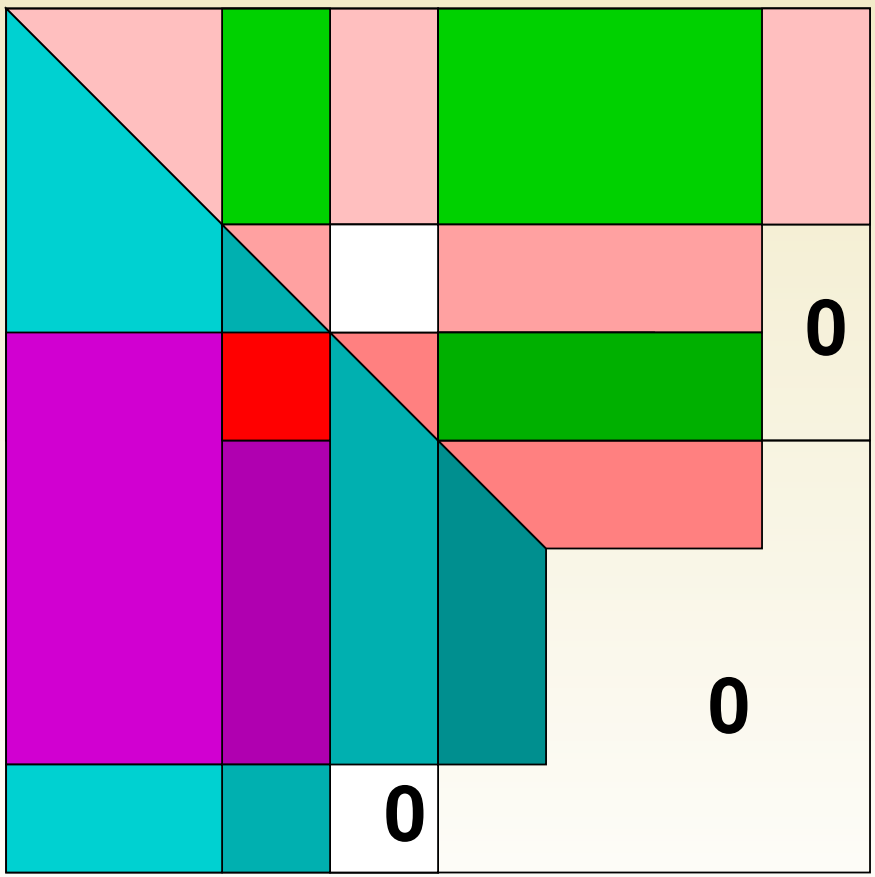
- Recursive cutting according to row and columns
- We can permut the blocks in a way that we can obtain row rank profile and column rank profile at the same time.
- rank profile of all leading sub-matrices



[Dumas, Pernet, Sultan, ISSAC 2013]

Algorithme PLUQ Quad-recursif

- Recursive cutting according to row and columns
- We can permut the blocks in a way that we can obtain row rank profile and column rank profile at the same time.
- rank profile of all leading sub-matrices



[Dumas, Pernet, Sultan, ISSAC 2013]

Outline

Conclusion

- Exact Computation
- Parallelization in exact
  - Trade-off : (Tiled, block) $<=>$ (adaptative, less sync.)
  - Specificity in Exact/Numeric : rank, rank profile
    $\rightarrow$ New issues and trade-off /Numeric $\&$ Parallel Numeric
- dataflow synchro. LUP :
  $\rightarrow$better adaptatibity
  $\rightarrow$ more parallelism
- PLUQ
  - Dynamic scheduling CUP :
    $\rightarrow$ dynamic block size, parallelism ?
  - new algorithm to parallelize : recursive, tile ?

Thank you for your attention !