Adaptive and Generic Parallel Exact Linear Algebra

Ziad SULTAN

Université Grenoble Aples

PhD Defense

June 17, 2016

Mme Laura Grigori (Referee)
M. Arne Stojohann (Referee)
M. Pascal Giorgi (Examiner)
M. Denis Trystram (Examiner)
M. Jean-Guillaume Dumas (Director)
M. Clément Pernet (Co-Director)

PhD research context

- Parallel computing



Symmetric Multi-Processors

PhD research context

- Parallel computing

Symmetric Multi-Processors

PhD research context

- Parallel computing

Symmetric Multi-Processors

- Effective exact parallel linear alegbra
  - Solve target problems : dedicated codes
  - Widely distributed software : general purpose codes (SAGE, Macauley2)

PhD research context

- Parallel computing



Symmetric Multi-Processors

- Effective exact parallel linear alegbra
  - Solve target problems : dedicated codes
  - Widely distributed software : general purpose codes (SAGE, Macauley2)
- Design a software for parallel exact linear algebra

Exact linear algebra

Exact computation

- Computation in computer algebra
  $\rightarrow$ computing exactly : over $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}[x]$
- In practice, often boils down to computation over prime fields $\mathbb{Z}/p\mathbb{Z}$

Exact linear algebra

Exact computation

- Computation in computer algebra
  $\to$ computing exactly : over $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}[x]$
- In practice, often boils down to computation over prime fields $\mathbb{Z}/p\mathbb{Z}$

Exact linear algebra applications

- Breaking Discrete Log Pb. in quasi-polynomial time [Barbulescu & al.14]
- Building modular form databases to test the BSD conjecture [Stein 12]
- Exact mixed-integer programming [Steffy et al. 12]
- Formal verification of Hales proof of Kepler conjecture [Hales 05]

Use case example of an application

HPAC on-going Challenge : D.L.P. cryptanalysis over curves over $\mathbb{F}(2^{29})$.

Problem dimensions

- Sparse matrix with 126M var. / 130M eq.
- Modulo a prime number on 114 bits : 20769187434139310549529495610151239
- Matrix has 520M non-zero

Use case example of an application

HPAC on-going Challenge : D.L.P. cryptanalysis over curves over $\mathbb{F}(2^{29})$.

Problem dimensions

- Sparse matrix with 126M var. / 130M eq.
- Modulo a prime number on 114 bits :
  20769187434139310549529495610151239
- Matrix has 520M non-zero

Main steps of block Wiedemann

- First filtering (structured Gauss)
  $\rightarrow$ nRows : 8.7M, nCols : 8.7M.
  $\rightarrow$ Matrix has 810M non-zero with blocs 32×16
- MinPoly coefficients 16×16, degree 545966
  $\rightarrow$ needs efficient PLUQ factorization !
- Evaluation uses M.M. : $(n \times 32)$ times (32×32) $\rightarrow$ n is large !

Dense exact linear algebra

Dense linear algebra : A key building block for :

- dense problems by nature (Hermite-Padé approx, ...)
- Sparse problems degenerate to dense :
  - Sparse Direct :
    Switch to dense after fill-in
  - Sparse Iterative :
    Induce dense elimination on blocks of iterated vectors
    (block-Wiedemann, block Lanczos, ...)

Gaussian elimination in exact dense algebra

Gaussian elimination is a building block in dense linear algebra

Matrix factorization (**LU decomposition**)

- Solving linear systems
- Computing determinant
- Rank.

Linear dependencies (**Echelon structure**)

- Characteristic Polynomial : Finding Krylov basis [Keller Gehrig 85]
- Grobner basis computation : F4 algorithm [FGB]

Design of parallel dense exact linear algebra

	numerical	exact
Sequential	BLAS, LAPACK	FFLAS-FFPACK
Parallel	pBLAS, ScaLAPACK	

Design of parallel dense exact linear algebra

	numerical	exact
Sequential	BLAS, LAPACK	FFLAS-FFPACK
Parallel	pBLAS, ScaLAPACK	this work

Design of parallel dense exact linear algebra

	numerical	exact
Sequential	BLAS, LAPACK	FFLAS-FFPACK
Parallel	pBLAS, ScaLAPACK	this work

Parallelizing dense linear algebra

- Specificities of exact linear algebra
  - Recursive algorithms
  - Rank deficiencies
- Similarities with numerical linear algebra
  Parallel blocking is constrained by pivoting :

  Numerical : ensuring numerical stability
  Exact : recovering rank profiles and echelon structure

Outline

Outline

1. Pivoting and rank profiles
2. Generic parallel Linear Algebra
3. Parallel exact Gaussian elimination

Linear dependencies and row/column rank profiles

Definition (Row Rank Profile : RowRP)

Given $A \in K^{m \times n}$, $r =$ *rank*(A).

informally : *first* r *linearly independent rows*

formally : *lexico-minimal sub-sequence of* $(1, \ldots, m)$ *of* r *indices of linearly independant rows.*

Example

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Linear dependencies and row/column rank profiles

Definition (Row Rank Profile : RowRP)

Given $A \in K^{m\times n}, r = \text{rank}(A)$.

informally : *first* r *linearly independent rows*

formally : *lexico-minimal sub-sequence of* $(1, \ldots, m)$ *of* r *indices of linearly independant rows.*

Example

$$\begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 0 & 1 & 0 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} \end{bmatrix}$$

Rank = 3
RowRP = $\{$ *1,2,4* $\}$

Linear dependencies and row/column rank profiles

Definition (Column Rank Profile : ColRP)

Given $A \in K^{m\times n}$, $r = \mathit{rank}(A)$.

informally : *first* r *linearly independent columns*

formally : *lexico-minimal sub-sequence of* $(1,\ldots,m)$ *of* r *indices of linearly independant columns.*

Example

$$\begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{0} & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & 0 \end{bmatrix}$$

Rank = 3
RowRP = $\{1,2,4\}$
ColRP = $\{1,2,3\}$

Linear dependencies and row/column rank profiles

Definition (Column Rank Profile : ColRP)

Given $A \in K^{m\times n}, r = \text{rank}(A)$.

informally : *first r linearly independent columns*

formally : *lexico-minimal sub-sequence of $(1,\ldots,m)$ of r indices of linearly independant columns.*

Example

$$\begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{0} & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & 0 \end{bmatrix}$$

Rank = 3
RowRP = {1,2,4}
ColRP = {1,2,3}

Generic RowRP/ColRP : if it equals $\{1,\ldots,r\}$.

Linear dependencies and row/column rank profiles

Definition (Column Rank Profile : ColRP)

Given $A \in K^{m\times n}, r = \text{rank}(A)$.

informally : *first* r *linearly independent columns*

formally : *lexico-minimal sub-sequence of* $(1,\ldots,m)$ *of* r *indices of linearly independant columns.*

Example

$$\begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{0} & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & 0 \end{bmatrix}$$

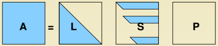
Rank = 3
RowRP = $\{$*1,2,4*$\}$
ColRP = $\{$*1,2,3*$\}$ $\rightarrow$ *Generic ColRP.*

Generic RowRP/ColRP : if it equals $\{1,\ldots,r\}$.

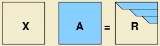
Computing rank profiles

Via Gaussian elimination revealing row echelon forms :

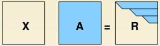
[Ibarra, Moran and Hui 82]



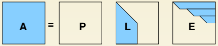
[Keller-Gehrig 85]



[Storjohann 00]



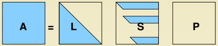
[Jeannerod, Pernet and Storjohann 13]



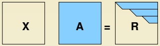
Computing rank profiles

Via Gaussian elimination revealing row echelon forms :

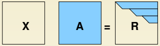
[Ibarra, Moran and Hui 82]



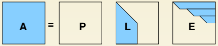
[Keller-Gehrig 85]



[Storjohann 00]



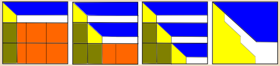
[Jeannerod, Pernet and Storjohann 13]



Lessons learned (or what we thought was necessary) :

- treat rows in order
- exhaust all columns before next row
- **slab** block splitting (rec or iter)

$\Rightarrow$ similar to partial pivoting



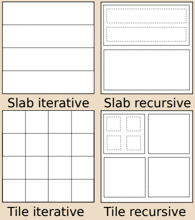
Motivation

Need more flexible blocking

Slab blocking

- can lead to inefficient memory access patterns
- is harder to parallelize

Tile blocking instead ?



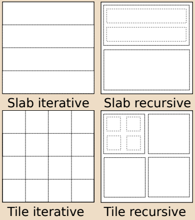
Motivation

Need more flexible blocking

Slab blocking

- can lead to inefficient memory access patterns
- is harder to parallelize

Tile blocking instead ?



Gathering linear independence invariants

Two ways to look at a matrix (looking left or right) :

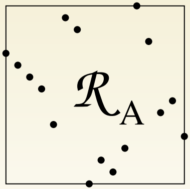
- Row rank profile, column echelon form
- Column rank profile, row echelon form

Unique invariant ?

The rank profile Matrix

Theorem

Let $A \in \mathrm{F}^{m \times n}$.
There exists a unique, $m \times n$, *rank*(A)*-sub-permutation matrix* $\mathcal{R}_A$ *of which every leading sub-matrix has the same rank as the corresponding leading sub-matrix of* A.



The rank profile Matrix

Theorem

Let $A \in \mathrm{F}^{m\times n}$.
There exists a unique, $m \times n$, rank(A)-sub-permutation matrix $\mathcal{R}_A$ of which every leading sub-matrix has the same rank as the corresponding leading sub-matrix of A.

Example

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 1 & 3 & 2 & 0 \\ 2 & 5 & 4 & 7 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The rank profile Matrix

Theorem

Let $A \in \mathrm{F}^{m \times n}$.
There exists a unique, $m \times n$, rank(A)-sub-permutation matrix $\mathcal{R}_A$ of which every leading sub-matrix has the same rank as the corresponding leading sub-matrix of A.

Example

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 1 & 3 & 2 & 0 \\ 2 & 5 & 4 & 7 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The rank profile Matrix

Theorem

Let $A \in \mathrm{F}^{m\times n}$.
There exists a unique, $m \times n$, rank(A)-sub-permutation matrix $\mathcal{R}_A$ of which every leading sub-matrix has the same rank as the corresponding leading sub-matrix of A.

Example

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 1 & 3 & 2 & 0 \\ 2 & 5 & 4 & 7 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The rank profile Matrix

Theorem

Let $A \in F^{m\times n}$.
There exists a unique, $m \times n$, rank(A)-sub-permutation matrix $\mathcal{R}_A$ of which every leading sub-matrix has the same rank as the corresponding leading sub-matrix of A.

Example

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 1 & 3 & 2 & 0 \\ 2 & 5 & 4 & 7 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The rank profile Matrix

Theorem

Let $A \in \mathrm{F}^{m \times n}$.
There exists a unique, $m \times n$, rank(A)-sub-permutation matrix $\mathcal{R}_A$ of which every leading sub-matrix has the same rank as the corresponding leading sub-matrix of A.

Example

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 1 & 3 & 2 & 0 \\ 2 & 5 & 4 & 7 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Properties of the rank profile matrix

Particular cases

- A invertible $\Leftrightarrow \mathcal{R}_A$ is a permutation
- A is square with generic rank profile $\Leftrightarrow \mathcal{R}_A = I_n$

Properties of the rank profile matrix

Particular cases

- A invertible $\Leftrightarrow \mathcal{R}_A$ is a permutation
- A is square with generic rank profile $\Leftrightarrow \mathcal{R}_A = I_n$

Properties

- $\mathcal{R}_A$ encodes the $RowRP(A)$ and the $ColRP(A)$
- All leading rank profiles
- $\mathcal{R}_A$ is unique $\implies$ new normal form.

When does a PLUQ decomposition reveal the rank profile matrix ?

Focus on the pivoting strategy :

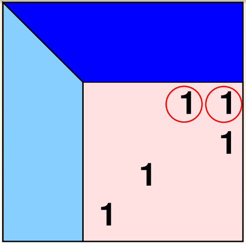
- Pivot search :
  - finding a pivot with minimal coordinates
- Permutation to bring the pivot to the main diagonal

Pivoting and permutation strategies

Pivot Search

Pivot's (i, j) position minimizes some pre-order :

Row order : any non-zero on the first non-zero row

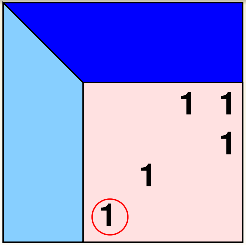


Pivoting and permutation strategies

Pivot Search

Pivot's (i,j) position minimizes some pre-order :

Row/Col order : any non-zero on the first non-zero row/col



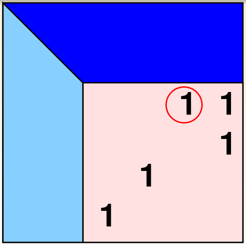
Pivoting and permutation strategies

Pivot Search

Pivot's (i, j) position minimizes some pre-order :

Row/Col order : any non-zero on the first non-zero row/col

Lex order : first non-zero on the first non-zero row



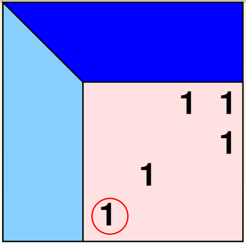
Pivoting and permutation strategies

Pivot Search

Pivot's (i,j) position minimizes some pre-order :

Row/Col order : any non-zero on the first non-zero row/col

Lex/RevLex order : first non-zero on the first non-zero row/col



Pivoting and permutation strategies

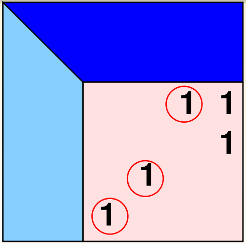
Pivot Search

Pivot's (i,j) position minimizes some pre-order :

Row/Col order : any non-zero on the first non-zero row/col

Lex/RevLex order : first non-zero on the first non-zero row/col

Product order : first non-zero in the (i,j) leading sub-matrix



Pivoting and permutation strategies

Pivot Search

Pivot's (i, j) position minimizes some pre-order :

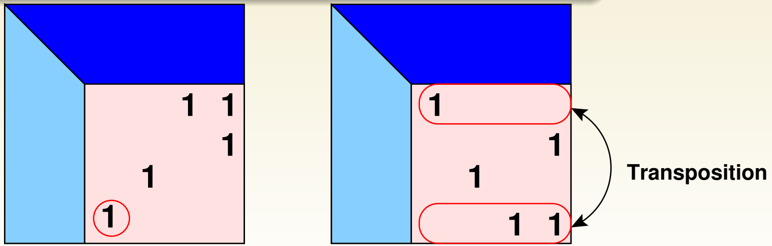
Row/Col order : any non-zero on the first non-zero row/col

Lex/RevLex order : first non-zero on the first non-zero row/col

Product order : first non-zero in the (i, j) leading sub-matrix

Permutation

- Transpositions



Pivoting and permutation strategies

Pivot Search

Pivot's (i,j) position minimizes some pre-order :

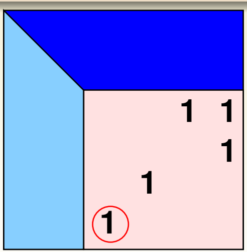
Row/Col order : any non-zero on the first non-zero row/col

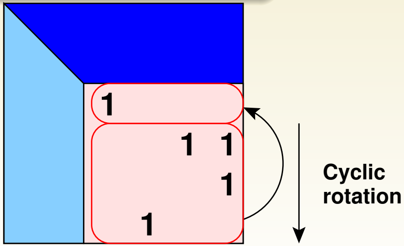
Lex/RevLex order : first non-zero on the first non-zero row/col

Product order : first non-zero in the (i,j) leading sub-matrix

Permutation

- Transpositions
- Cyclic Rotations





Pivoting strategies revealing rank profiles

Search	**Row perm.**	**Col. perm.**	RowRP	ColRP	**Instance**
Row order Col. order					
Lexico.					
Rev. lex.					
Product					

Pivoting strategies revealing rank profiles

Search	**Row perm.**	**Col. perm.**	RowRP	ColRP	**Instance**
Row order Col. order	Transposition	Transposition	✓		[IMH82] [JPS13]
Lexico.					
Rev. lex.					
Product					

Pivoting strategies revealing rank profiles

Search	Row perm.	Col. perm.	RowRP	ColRP	Instance
Row order	Transposition	Transposition	✓		[IMH82] [JPS13]
Col. order	Transposition	Transposition		✓	[KG85] [JPS13]
Lexico.					
Rev. lex.					
Product					

Pivoting strategies revealing rank profiles

Search	**Row perm.**	**Col. perm.**	RowRP	ColRP	**Instance**
Row order	Transposition	Transposition	✓		[IMH82] [JPS13]
Col. order	Transposition	Transposition		✓	[KG85] [JPS13]
Lexico.	Transposition	Transposition	✓		[Sto00]
Rev. lex.	Transposition	Transposition		✓	[Sto00]
Product					

Pivoting strategies revealing rank profiles

Search	**Row perm.**	**Col. perm.**	RowRP	ColRP	$\mathcal{R}_A$	**Instance**
Row order	Transposition	Transposition	✓			[IMH82] [JPS13]
Col. order	Transposition	Transposition		✓		[KG85] [JPS13]
Lexico.	Transposition	Transposition	✓			[Sto00]
Rev. lex.	Transposition	Transposition		✓		[Sto00]
Product	Rotation	Rotation	✓	✓	✓	[DPS13]

$P, L, U, Q \leftarrow PLUQ(A)$ and $P \begin{bmatrix} I_r & \\ & 0 \end{bmatrix} Q = \mathcal{R}_A$.

Pivoting strategies revealing rank profiles

Search	**Row perm.**	**Col. perm.**	RowRP	ColRP	$\mathcal{R}_A$	**Instance**
Row order	Transposition	Transposition	✓			[IMH82] [JPS13]
Col. order	Transposition	Transposition		✓		[KG85] [JPS13]
Lexico.	Transposition	Transposition	✓			[Sto00]
Rev. lex.	Transposition	Transposition		✓		[Sto00]
Product	Rotation	Transposition	✓			[DPS15]
Product	Transposition	Rotation		✓		[DPS15]
Product	Rotation	Rotation	✓	✓	✓	[DPS13]

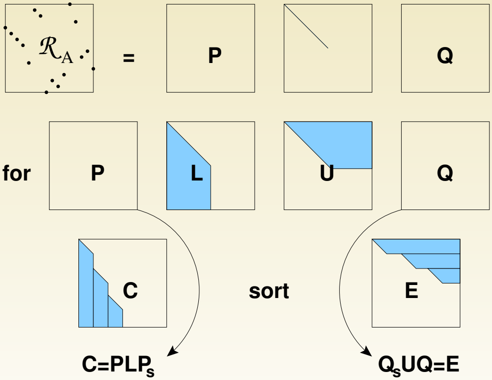
$P, L, U, Q \leftarrow PLUQ(A)$ and $P \begin{bmatrix} I_r & \\ & 0 \end{bmatrix} Q = \mathcal{R}_A$.

Pivoting strategies revealing rank profiles

Search	**Row perm.**	**Col. perm.**	RowRP	ColRP	$\mathcal{R}_A$	**Instance**
Row order	Transposition	Transposition	✓			[IMH82] [JPS13]
Col. order	Transposition	Transposition		✓		[KG85] [JPS13]
Lexico.	Transposition	Transposition	✓			[Sto00]
Lexico.	Transposition	Rotation	✓	✓	✓	[DPS15]
Lexico.	Rotation	Rotation	✓	✓	✓	[DPS15]
Rev. lex.	Transposition	Transposition		✓		[Sto00]
Rev. lex.	Rotation	Transposition	✓	✓	✓	[DPS15]
Rev. lex.	Rotation	Rotation	✓	✓	✓	[DPS15]
Product	Rotation	Transposition	✓			[DPS15]
Product	Transposition	Rotation		✓		[DPS15]
Product	Rotation	Rotation	✓	✓	✓	[DPS13]

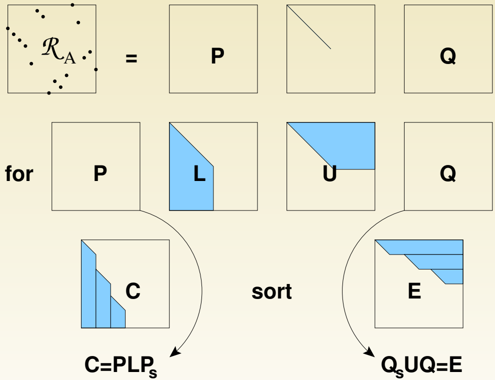
$P, L, U, Q \leftarrow PLUQ(A)$ and $P \begin{bmatrix} I_r & \\ & 0 \end{bmatrix} Q = \mathcal{R}_A$.

Echelon forms



$$C = P \left[L.P_s \;\; 0_{m\times(n-r)} \right] \qquad E = \begin{bmatrix} Q_s.U \\ 0_{(n-r)\times n} \end{bmatrix} Q$$

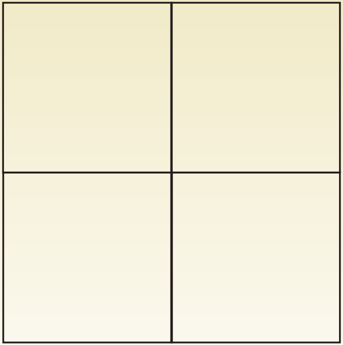
Echelon forms



$$C = P\left[\begin{smallmatrix} L.P_s & 0_{m\times(n-r)}\end{smallmatrix}\right],\ F = P_s^T Q_s^T,\ E = \left[\begin{smallmatrix} Q_s.U \\ 0_{(n-r)\times n}\end{smallmatrix}\right] Q$$

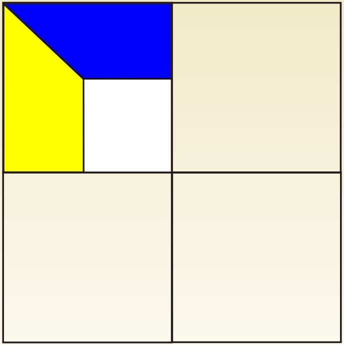
Bonus : Generalized Bruhat *CFE*.

Tile recursive PLUQ algorithm



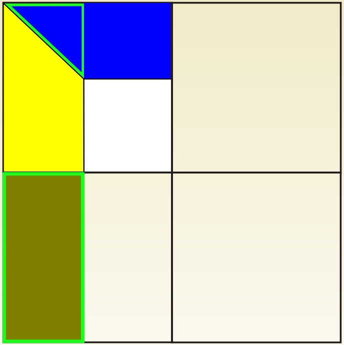
2×2 block splitting

Tile recursive PLUQ algorithm



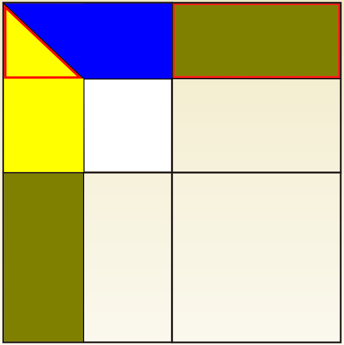
Recursive call

Tile recursive PLUQ algorithm



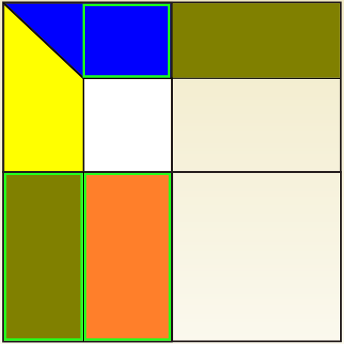
$\texttt{TRSM} : B \leftarrow BU^{-1}$

Tile recursive PLUQ algorithm



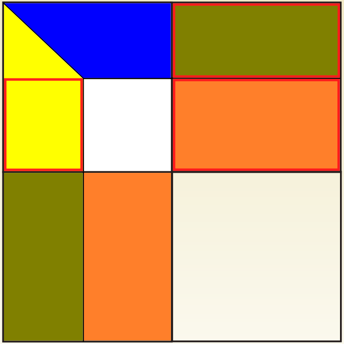
$\texttt{TRSM} : B \leftarrow L^{-1}B$

Tile recursive PLUQ algorithm



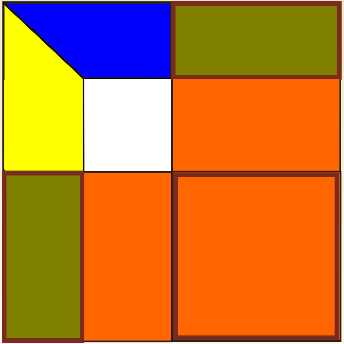
$\texttt{fgemm} : C \leftarrow C - A \times B$

Tile recursive PLUQ algorithm



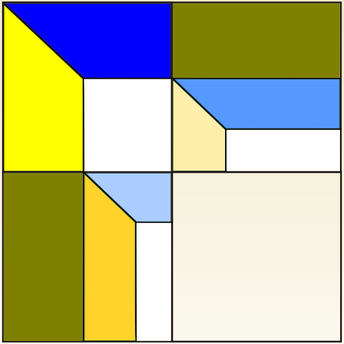
$\texttt{fgemm} : C \leftarrow C - A \times B$

Tile recursive PLUQ algorithm



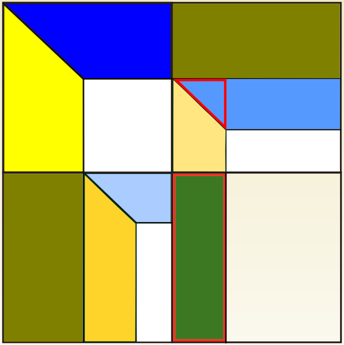
$\texttt{fgemm} : C \leftarrow C - A \times B$

Tile recursive PLUQ algorithm



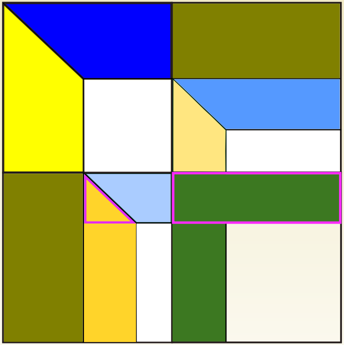
2 independent recursive calls (product order search)

Tile recursive PLUQ algorithm



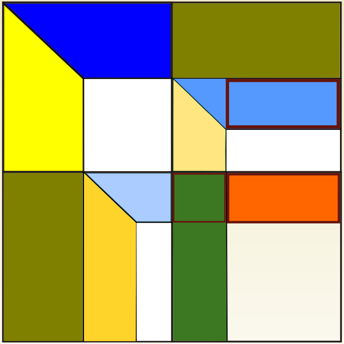
$\texttt{TRSM} : B \leftarrow BU^{-1}$

Tile recursive PLUQ algorithm



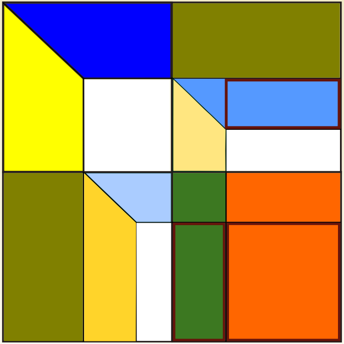
$\texttt{TRSM} : B \leftarrow L^{-1}B$

Tile recursive PLUQ algorithm



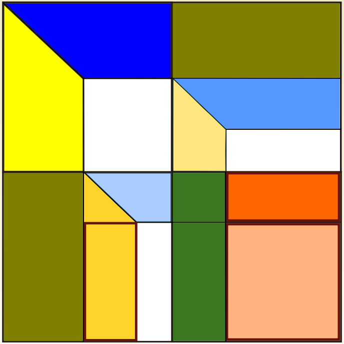
$\texttt{fgemm} : C \leftarrow C - A \times B$

Tile recursive PLUQ algorithm



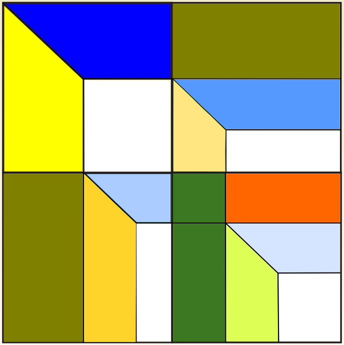
$\texttt{fgemm} : C \leftarrow C - A \times B$

Tile recursive PLUQ algorithm



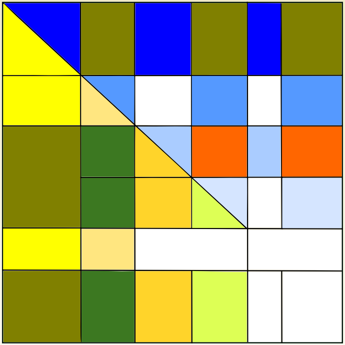
$\texttt{fgemm} : C \leftarrow C - A \times B$

Tile recursive PLUQ algorithm



Recursive call

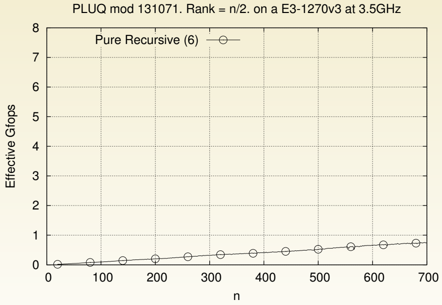
Tile recursive PLUQ algorithm



Puzzle game (block permutations)

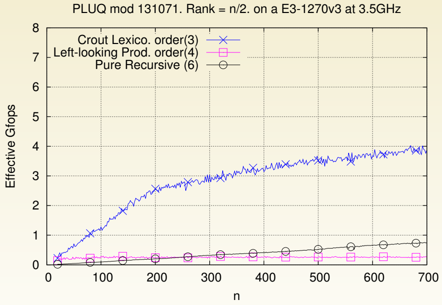
New PLUQ algorithm

- New state of the art algo that computes faster PLUQ decomposition
- Computes more information (the rank profile matrix $\mathcal{R}_A$)



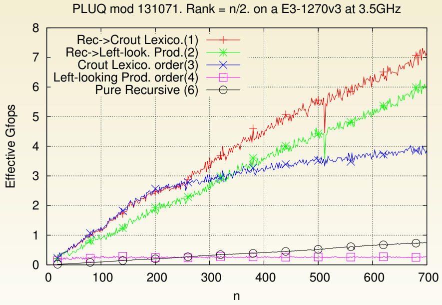
New PLUQ algorithm

- New state of the art algo that computes faster PLUQ decomposition
- Computes more information (the rank profile matrix $\mathcal{R}_A$)



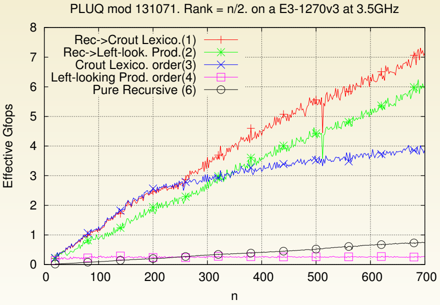
New PLUQ algorithm

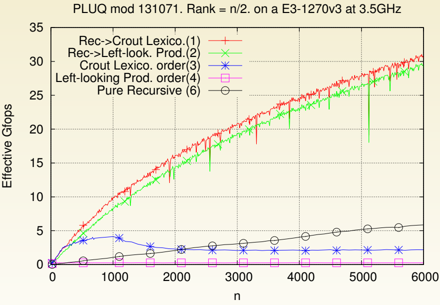
- New state of the art algo that computes faster PLUQ decomposition
- Computes more information (the rank profile matrix $\mathcal{R}_A$)



New PLUQ algorithm

- New state of the art algo that computes faster PLUQ decomposition
- Computes more information (the rank profile matrix $\mathcal{R}_A$)





Execution on 1 core (3.5GHz) $\rightarrow$ effective 31 Gfops (AVX2 + sub-cubic complexity)

Outline

FFLAS-FFPACK library

FFLAS-FFPACK features

- High performance implementation of basic linear algebra routines over word size prime fields
- Exact alternative to the numerical BLAS library
- Exact triangularization, Sys. solving, Det, Inv., CharPoly

Parallel FFLAS-FFPACK

Explore :

- several algorithms and variants
- parallel runtimes and languages :
  - unified parallel language harnessing different runtimes (OMP, TBB, xKaapi, ...)
  - Abstraction for the user
- data parallelism vs task parallelism

Parallel computation constraints : exact and numeric

In state of the art numerical libraries :

- Often non singular matrices with fixed static cutting.
  $\rightarrow$ easier to manually map and schedule tasks or threads.
- Use of iterative algorithms $\rightarrow$ often one or two levels of parallelism.

Parallel computation constraints : exact and numeric

In state of the art numerical libraries :

- Often non singular matrices with fixed static cutting.
  $\rightarrow$ easier to manually map and schedule tasks or threads.
- Use of iterative algorithms $\rightarrow$ often one or two levels of parallelism.

Our experience in exact linear algebra :

- Sub-cubic complexity : $O(n^\omega)$ [Strassen]
  $\rightarrow$ Coarser grain cutting
  $\rightarrow$ Recursive algorithms.
  $\rightarrow$ Parallel runtime system that implements well recursive tasks.
- Rank deficiencies $\rightarrow$ tasks of unbalanced workloads.
- Recursion and code composition $\rightarrow$ multiple levels of parallelism.

Parallel computation constraints : exact and numeric

In state of the art numerical libraries :

- Often non singular matrices with fixed static cutting.
  $\rightarrow$ easier to manually map and schedule tasks or threads.
- Use of iterative algorithms $\rightarrow$ often one or two levels of parallelism.

Our experience in exact linear algebra :

- Sub-cubic complexity : $O(n^\omega)$ [Strassen]
  $\rightarrow$ Coarser grain cutting
  $\rightarrow$ Recursive algorithms.
  $\rightarrow$ Parallel runtime system that implements well recursive tasks.
- Rank deficiencies $\rightarrow$ tasks of unbalanced workloads.
- Recursion and code composition $\rightarrow$ multiple levels of parallelism.

$\rightarrow$ Need for a high level parallel programming environments

Requirements of high level parallel programming environments

Features required

Portability, Performance and scalability. But more precisely :

- Runtime system with good performance for recursive tasks.
- Handle efficiently unbalanced workloads.
- Efficient range cutting for parallel for.

Requirements of high level parallel programming environments

Features required

Portability, Performance and scalability. But more precisely :

- Runtime system with good performance for recursive tasks.
- Handle efficiently unbalanced workloads.
- Efficient range cutting for parallel for.

No parallel environment offers all these features
$\rightarrow$ Need to design a code independently from the runtime system
$\rightarrow$ Using runtime systems as a plugin

Runtime systems to be supported

OpenMP3.x and 4.0 supported directives : (using libgomp)

- Data sharing attributes :
  - OMP3 `shared` : data visible and accessible by all threads
  - OMP3 `firstprivate` : local copy of original value
  - OMP4 `depend` : set data dependencies
- Synchronization clauses : `#pragma omp taskwait`

xKaapi : via the libkomp [BDG12] library :

- OpenMP directives → xKaapi tasks.
- Re-implem. of task handling and management.
- Better recursive tasks execution.

TBB : designed for nested and recursive parallelism

- `parallel_for`
- `tbb::task_group, wait(), run()` using C++11 lambda functions

PALADIn

Parallel **A**lgebraic **L**inear **A**lgebra **D**edicated **In**terface

Mainly macro-based keywords

- No function call runtime overhead when using macros.
- No important modifications to be done to original program.
- Macros can be used also for C-based libraries.

Complementary C++ template functions

- Implement the different cutting strategies.
- Store the iterators

PALADIn description : data parallelism

Data parallelism : SPMD programming

- Parallel region : chunks are dispatched on multiple proc.
- Supported : `PARFOR1D`, `PARFOR2D`, `PARFORBLOCK1D`, `PARFORBLOCK2D`.

Example : Loop Summing in C++

```
for(size_t i=0; i<n; ++i){
 T[i] = T1[i] + T2[i];
}
```

Example : Loop Summing in OpenMP

```
#pragma omp parallel for
for(size_t i=0; i<n; ++i){
 T[i] = T1[i] + T2[i];
}
```

Example : Loop Summing in PALADIn

```
PARFOR1D(i, n, SPLITTER(),
         T[i] = T1[i]+T2[i];
        );
```

→ The `SPLITTER` keyword sets the cutting strategy.

Iterative Cutting Strategies 1D

Splitting over one dimension

- `SPLITTER(`p`, THREADS)` : p partitions $= \#tasks$
- `SPLITTER(`p`, GRAIN)` : BlockSize : $BS = p$
- `SPLITTER(`p`, FIXED)` : BlockSize : $BS = 256$
- `SPLITTER(`p`)` : p tasks with default strategy (`THREADS`)
- `SPLITTER()` : default strategy with $p = \#$ available processors

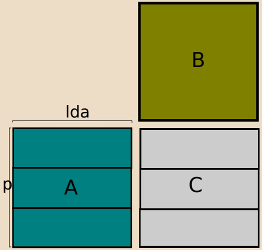
Code example : Matrix add in parallel

```
void pfadd(const Field & F,const Element *A,const Element *B, Element *C, size_t n){
   PARFORBLOCK1D(it , n, SPLITTER(32, THREADS),
            FFLAS::fadd(F, it.end()-it.begin(), n,
                        A+it.begin()*n, n,
                        B+it.begin()*n, n,
                        C+it.begin()*n, n);
            );
 }
```

Iterative cutting strategies 2D

Data parallelism : SPLITTER keyword

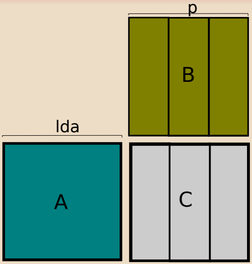
- `SPLITTER(`p`, ROW, THREADS)` : p row blocks
- `SPLITTER(`p`, ROW, FIXED)` : row $BS = 256$
- `SPLITTER(`p`, ROW, GRAIN)` : row $BS = p$



Iterative cutting strategies 2D

Data parallelism : SPLITTER keyword

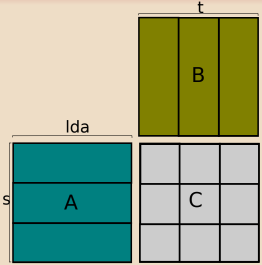
- `SPLITTER(`p`, ROW, THREADS)` : p row blocks
- `SPLITTER(`p`, ROW, FIXED)` : row $BS = 256$
- `SPLITTER(`p`, ROW, GRAIN)` : row $BS = p$
- `SPLITTER(`p`, COLUMN, THREADS)` : p col blocks
- `SPLITTER(`p`, COLUMN, FIXED)` : col $BS = 256$
- `SPLITTER(`p`, COLUMN, GRAIN)` : col $BS = p$



Iterative cutting strategies 2D

Data parallelism : SPLITTER keyword

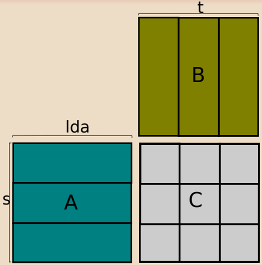
- `SPLITTER(`p`, ROW, THREADS)` : p row blocks
- `SPLITTER(`p`, ROW, FIXED)` : row $BS = 256$
- `SPLITTER(`p`, ROW, GRAIN)` : row $BS = p$
- `SPLITTER(`p`, COLUMN, THREADS)` : p col blocks
- `SPLITTER(`p`, COLUMN, FIXED)` : col $BS = 256$
- `SPLITTER(`p`, COLUMN, GRAIN)` : col $BS = p$
- `SPLITTER(`p`, BLOCK, THREADS)` : $s \times t$ blocks
- `SPLITTER(`p`, BLOCK, FIXED)` : $BS = 256$
- `SPLITTER(`p`, BLOCK, GRAIN)` : $BS = p$



Iterative cutting strategies 2D

Data parallelism : SPLITTER keyword

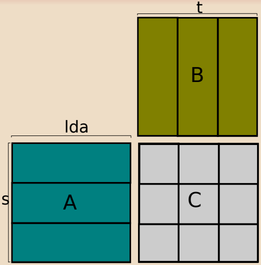
- `SPLITTER(`p`, ROW, THREADS)` : p row blocks
- `SPLITTER(`p`, ROW, FIXED)` : row $BS = 256$
- `SPLITTER(`p`, ROW, GRAIN)` : row $BS = p$
- `SPLITTER(`p`, COLUMN, THREADS)` : p col blocks
- `SPLITTER(`p`, COLUMN, FIXED)` : col $BS = 256$
- `SPLITTER(`p`, COLUMN, GRAIN)` : col $BS = p$
- `SPLITTER(`p`, BLOCK, THREADS)` : $s \times t$ blocks
- `SPLITTER(`p`, BLOCK, FIXED)` : $BS = 256$
- `SPLITTER(`p`, BLOCK, GRAIN)` : $BS = p$
- `NOSPLIT()` : sequential execution



Iterative cutting strategies 2D

Data parallelism : SPLITTER keyword

- `SPLITTER(`p`, ROW, THREADS)` : p row blocks
- `SPLITTER(`p`, ROW, FIXED)` : row $BS = 256$
- `SPLITTER(`p`, ROW, GRAIN)` : row $BS = p$
- `SPLITTER(`p`, COLUMN, THREADS)` : p col blocks
- `SPLITTER(`p`, COLUMN, FIXED)` : col $BS = 256$
- `SPLITTER(`p`, COLUMN, GRAIN)` : col $BS = p$
- `SPLITTER(`p`, BLOCK, THREADS)` : $s \times t$ blocks
- `SPLITTER(`p`, BLOCK, FIXED)` : $BS = 256$
- `SPLITTER(`p`, BLOCK, GRAIN)` : $BS = p$
- `NOSPLIT()` : sequential execution

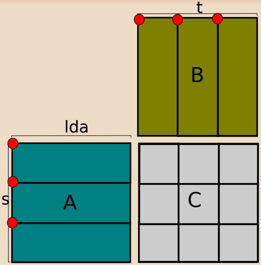



```
PARFORBLOCK2D(iter, m, n, SPLITTER(),
        fgemm( ..., A +iter.ibegin()*lda, lda,
                    B +iter.jbegin(), ldb, beta,
                    C +iter.ibegin()*ldc+iter.jbegin(), ldc);
      );
```

Iterative cutting strategies 2D

Data parallelism : SPLITTER keyword

- `SPLITTER(`p`, ROW, THREADS)` : p row blocks
- `SPLITTER(`p`, ROW, FIXED)` : row $BS = 256$
- `SPLITTER(`p`, ROW, GRAIN)` : row $BS = p$
- `SPLITTER(`p`, COLUMN, THREADS)` : p col blocks
- `SPLITTER(`p`, COLUMN, FIXED)` : col $BS = 256$
- `SPLITTER(`p`, COLUMN, GRAIN)` : col $BS = p$
- `SPLITTER(`p`, BLOCK, THREADS)` : $s \times t$ blocks
- `SPLITTER(`p`, BLOCK, FIXED)` : $BS = 256$
- `SPLITTER(`p`, BLOCK, GRAIN)` : $BS = p$
- `NOSPLIT()` : sequential execution

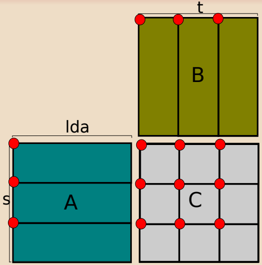



```
PARFORBLOCK2D(iter, m, n, SPLITTER(),
        fgemm( ..., A+iter.ibegin()*lda , lda,
                    B+iter.jbegin()     , ldb, beta,
                 C+iter.ibegin()*ldc+iter.jbegin(), ldc);
        );
```

Iterative cutting strategies 2D

Data parallelism : SPLITTER keyword

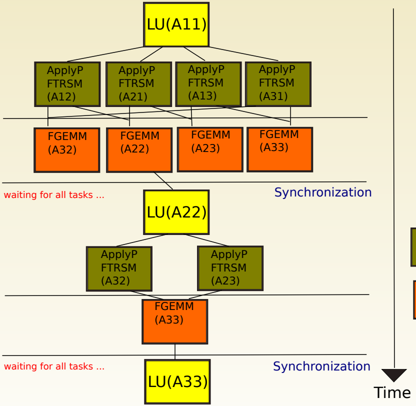
- `SPLITTER(p, ROW, THREADS)` : p row blocks
- `SPLITTER(p, ROW, FIXED)` : row $BS = 256$
- `SPLITTER(p, ROW, GRAIN)` : row $BS = p$
- `SPLITTER(p, COLUMN, THREADS)` : p col blocks
- `SPLITTER(p, COLUMN, FIXED)` : col $BS = 256$
- `SPLITTER(p, COLUMN, GRAIN)` : col $BS = p$
- `SPLITTER(p, BLOCK, THREADS)` : $s \times t$ blocks
- `SPLITTER(p, BLOCK, FIXED)` : $BS = 256$
- `SPLITTER(p, BLOCK, GRAIN)` : $BS = p$
- `NOSPLIT()` : sequential execution



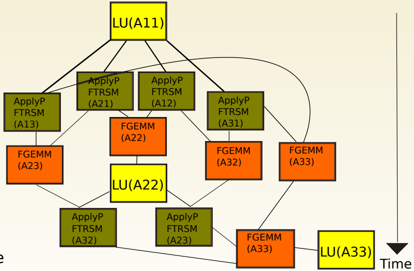

```
PARFORBLOCK2D(iter, m, n, SPLITTER(),
        fgemm( ..., A+iter.ibegin()*lda , lda,
                    B+iter.jbegin()     , ldb, beta,
                    C+iter.ibegin()*ldc+iter.jbegin() , ldc);
    );
```

Task parallelism

fork-join model :



data-flow model :



PALADIn description : task parallelism

Task parallelization : fork-join and dataflow models

- `PAR_BLOCK` : opens a parallel region.
- `SYNCH_GROUP` : Group of tasks synchronized upon exit.
- `TASK` : creates a task.
  - `REFERENCE(args...)` : specify variables captured by reference. By default all variables accessed by value.
  - `READ(args...)` : set var. that are read only.
  - `WRITE(args...)` : set var. that are written only.
  - `READWRITE(args...)` : set var. that are read then written.

PALADIn description : task parallelism

Task parallelization : fork-join and dataflow models

- `PAR_BLOCK` : opens a parallel region.
- `SYNCH_GROUP` : Group of tasks synchronized upon exit.
- `TASK` : creates a task.
  - `REFERENCE(args...)` : specify variables captured by reference. By default all variables accessed by value.
  - `READ(args...)` : set var. that are read only.
  - `WRITE(args...)` : set var. that are written only.
  - `READWRITE(args...)` : set var. that are read then written.

Example :

```
void axpy(const Element a, const Element b, Element &y){y += a*x;}
SYNCH_GROUP(
            TASK(MODE(READ(a,x) READWRITE(y)),
                 axpy(a,x,y));
           );
```

PALADIn description : task parallelism

Task parallelization : fork-join and dataflow models

- `PAR_BLOCK` : opens a parallel region.
- `SYNCH_GROUP` : Group of tasks synchronized upon exit.
- `TASK` : creates a task.
  - `REFERENCE(args...)` : specify variables captured by reference. By default all variables accessed by value.
  - `READ(args...)` : set var. that are read only.
  - `WRITE(args...)` : set var. that are written only.
  - `READWRITE(args...)` : set var. that are read then written.

Example :

```
void axpy(const Element a, const Element b, Element &y){y += a*x;}
SYNCH_GROUP(
            TASK(MODE(READ(a,x) READWRITE(y)),
                 axpy(a,x,y));
           );
```

Now we have a language to test our parallel exact linear algebra algorithms !

Parallel matrix multiplication cascading

Algorithms

- Classical algorithms : $O(n^3)$
- Fast algorithms : $O(n^\omega)$

Problem

What are the best possible cascades ?

Cascading

- Parallel classical variant switches to :
  - sequential fast
  - sequential classical
  - parallel fast
- iterative (BLOCK-THREADS)
- recursive (1D, 2D, 3D splitting)

Parallel matrix multiplication cascading

Algorithms

- Classical algorithms : $O(n^3)$
- Fast algorithms : $O(n^\omega)$

Problem

What are the best possible cascades ?

Cascading

- Parallel classical variant switches to :
  - sequential fast
  - sequential classical
  - parallel fast
- iterative (BLOCK-THREADS)
- recursive (1D, 2D, 3D splitting)
- Parallel fast variant switches to :
  - sequential fast
  - sequential classical
  - parallel classical
- recursive (Strassen-Winograd)

Parallel matrix multiplication cascading

Algorithms

- Classical algorithms : $O(n^3)$
- Fast algorithms : $O(n^\omega)$

Problem

What are the best possible cascades ?

Cascading

- Parallel classical variant switches to :
  - sequential fast
  - sequential classical
  - parallel fast
- Parallel fast variant switches to :
  - sequential fast
  - sequential classical
  - parallel classical

- iterative (BLOCK-THREADS)
- recursive (1D, 2D, 3D splitting)
- recursive (Strassen-Winograd)

Performance of pfgemm

pfgemm : Parallel classical variant → Sequential fast

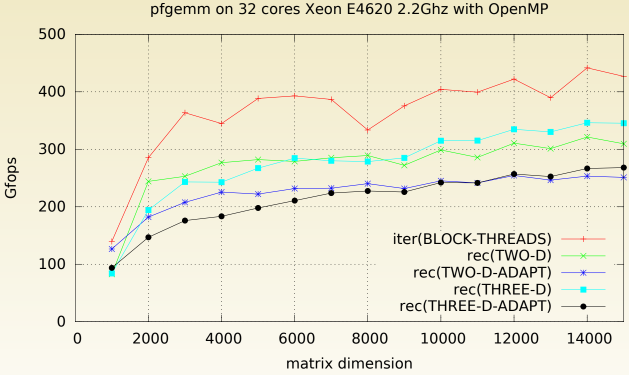


FIGURE : Speed of different matrix multiplication cutting strategies using OpenMP tasks

Performance of pfgemm

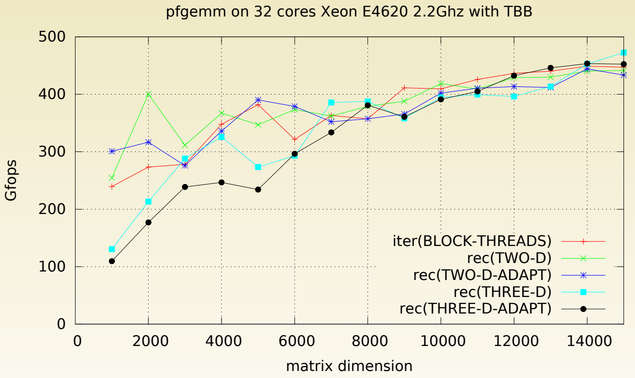


FIGURE : Speed of different matrix multiplication cutting strategies using TBB tasks

Performance of pfgemm

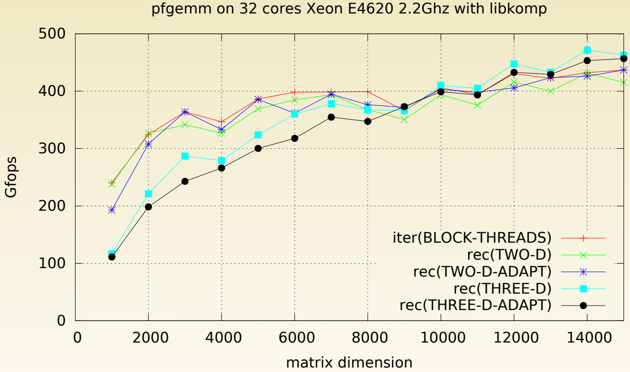
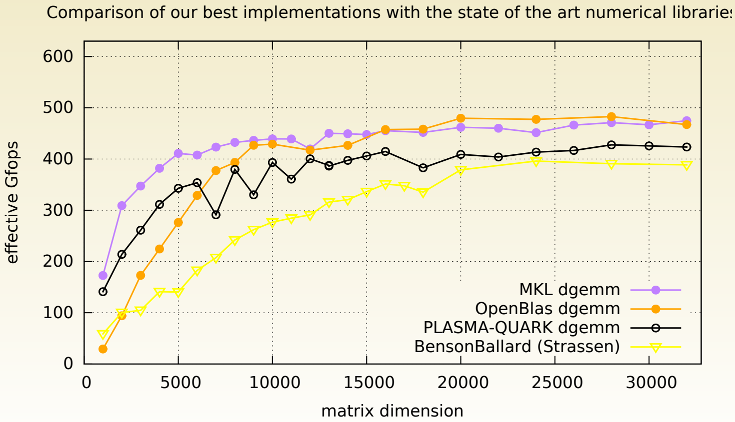


FIGURE : Speed of different matrix multiplication cutting strategies using xKaapi tasks

Parallel Matrix Multiplication : State of the art

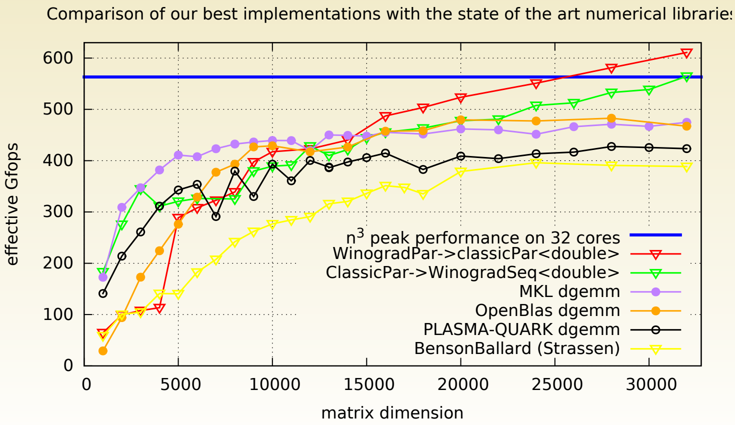
HPAC server : 32 cores Xeon E4620 2.2Ghz (4 NUMA sockets)



Parallel Matrix Multiplication : State of the art

HPAC server : 32 cores Xeon E4620 2.2Ghz (4 NUMA sockets)

Effective Gfops = $\dfrac{\text{\# of field ops using classic matrix product}}{\text{time}}$.



Outline

Gaussian elimination design

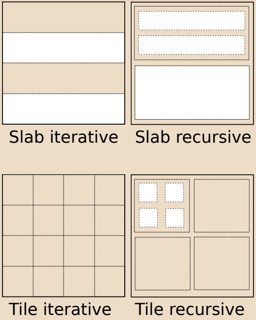
Reducing to MatMul : block versions

$\rightarrow$ Asymptotically faster ($O(n^\omega)$)
$\rightarrow$ Better cache efficiency

Variants of block versions

Split on one dimension :
$\rightarrow$ Row or Column slab cutting

Split on 2 dimensions :
$\rightarrow$ Tile cutting



Gaussian elimination design

Reducing to MatMul : block versions

$\rightarrow$ Asymptotically faster ($O(n^\omega)$)
$\rightarrow$ Better cache efficiency

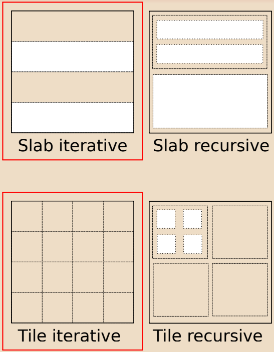
Variants of block versions

Iterative :

- Static $\rightarrow$ better data mapping control
- Dataflow parallel model $\rightarrow$ less sync

Recursive :

- Adaptive
- sub-cubic complexity
- No Dataflow $\rightarrow$ more sync



Gaussian elimination design

Reducing to MatMul : block versions

$\rightarrow$ Asymptotically faster ($O(n^\omega)$)
$\rightarrow$ Better cache efficiency

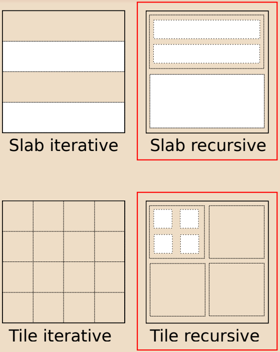
Variants of block versions

Iterative :

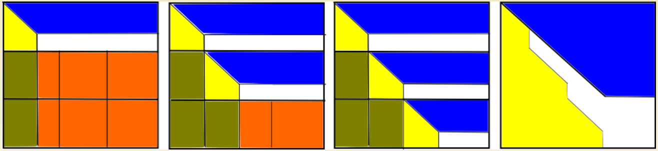
- Static $\rightarrow$ better data mapping control
- Dataflow parallel model $\rightarrow$ less sync

Recursive :

- Adaptive
- sub-cubic complexity
- No Dataflow $\rightarrow$ more sync



Slab iterative



Slab iterative

Expensive costly tasks in the critical path

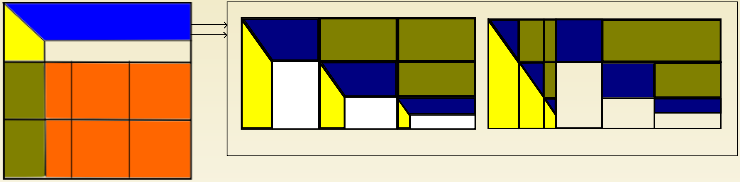
- Panel factorization in sequential

Rank dynamically revealed :

- Varying workload of each block op.

Tiled iterative PLUQ decomposition

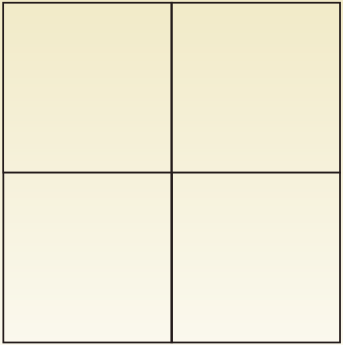
$\rightarrow$ Panel PLUQ decomposition on each slab



Slab iterative CUP to tile iterative PLUQ

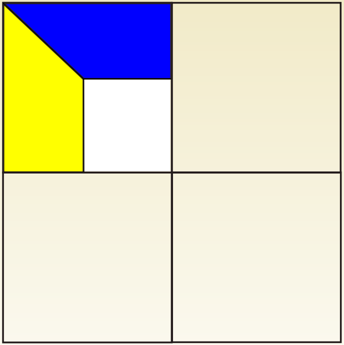
- Cutting according to columns
- Creating "more parallelism" : update tasks are concurrent
- Recovering rank profiles thanks to our pivoting strategies

Parallel tile recursive PLUQ algorithm



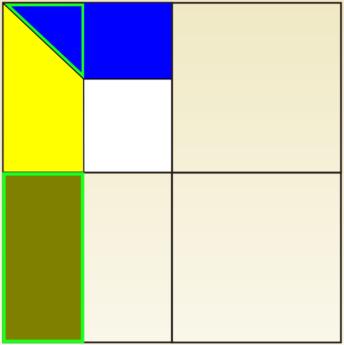
2×2 block splitting

Parallel tile recursive PLUQ algorithm



Recursive call

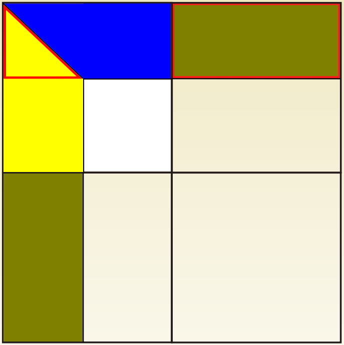
Parallel tile recursive PLUQ algorithm



$\mathrm{pTRSM} : B \leftarrow BU^{-1}$

```
TASK(MODE(READ(A) READWRITE(B)),
      pftrsm(..., A, lda, B, ldb));
```

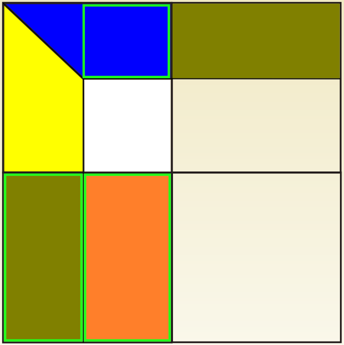
Parallel tile recursive PLUQ algorithm



$\mathrm{pTRSM} : B \leftarrow L^{-1}B$

```
TASK(MODE(READ(A) READWRITE(B)),
     pftrsm(..., A, lda, B, ldb));
```

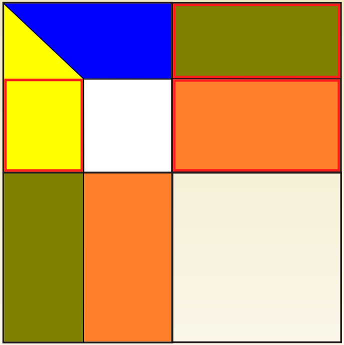
Parallel tile recursive PLUQ algorithm



$\texttt{pfgemm} : C \leftarrow C - A \times B$

```
TASK(MODE(READ(A,B) READWRITE(C)),
        pfgemm(..., A, lda, B, ldb));
```

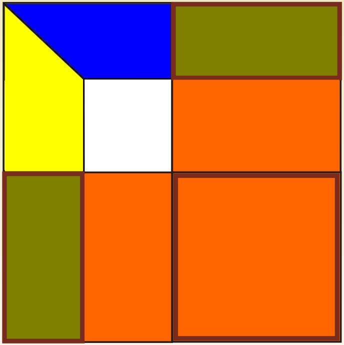
Parallel tile recursive PLUQ algorithm



$\texttt{pfgemm} : C \leftarrow C - A \times B$

```
TASK(MODE(READ(A,B) READWRITE(C)),
     pfgemm(..., A, lda, B, ldb));
```

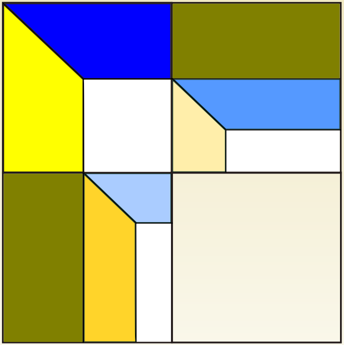
Parallel tile recursive PLUQ algorithm



$\texttt{pfgemm}: C \leftarrow C - A \times B$

```
TASK(MODE(READ(A,B) READWRITE(C)),
     pfgemm(..., A, lda, B, ldb));
```

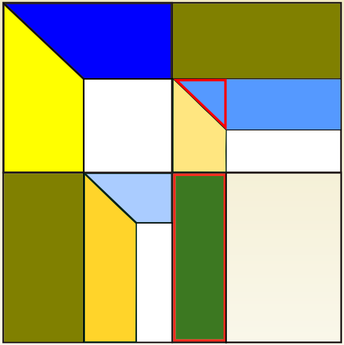
Parallel tile recursive PLUQ algorithm



2 independent recursive calls (concurrent → tasks)

```
TASK(MODE(READWRITE(A)),
     pluq(..., A, lda));
```

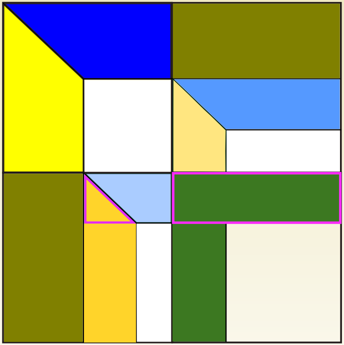
Parallel tile recursive PLUQ algorithm



$\mathtt{pTRSM}: B \leftarrow BU^{-1}$

```
TASK(MODE(READ(A) READWRITE(B)),
     pftrsm(..., A, lda, B, ldb));
```

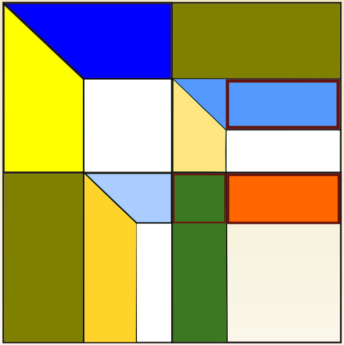
Parallel tile recursive PLUQ algorithm



$\texttt{pTRSM} : B \leftarrow L^{-1}B$

```
TASK(MODE(READ(A) READWRITE(B)),
      pftrsm(..., A, lda, B, ldb));
```

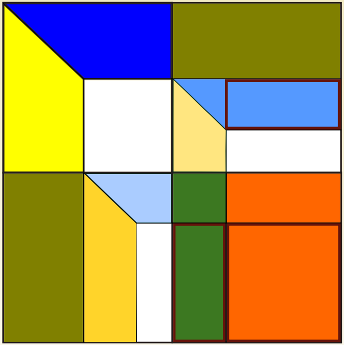
Parallel tile recursive PLUQ algorithm



$\texttt{pfgemm} : C \leftarrow C - A \times B$

```
TASK(MODE(READ(A,B) READWRITE(C)),
      pfgemm(..., A, lda, B, ldb));
```

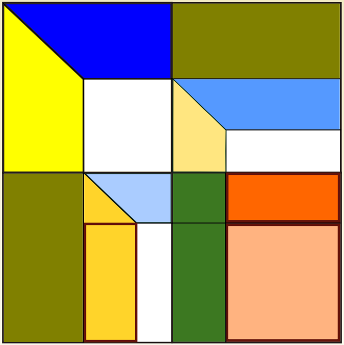
Parallel tile recursive PLUQ algorithm



$\texttt{pfgemm} : C \leftarrow C - A \times B$

```
TASK(MODE(READ(A,B) READWRITE(C)),
      pfgemm(..., A, lda, B, ldb));
```

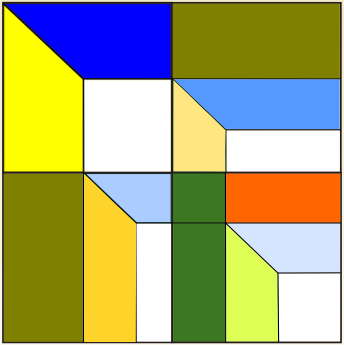
Parallel tile recursive PLUQ algorithm



$\texttt{pfgemm} : C \leftarrow C - A \times B$

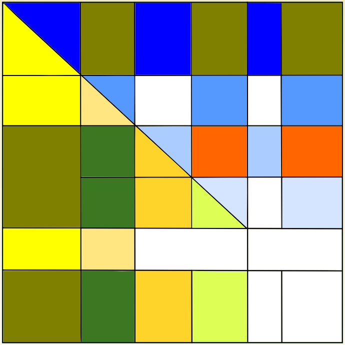
```
TASK(MODE(READ(A,B) READWRITE(C)),
     pfgemm(..., A, lda, B, ldb));
```

Parallel tile recursive PLUQ algorithm



Recursive call

Parallel tile recursive PLUQ algorithm

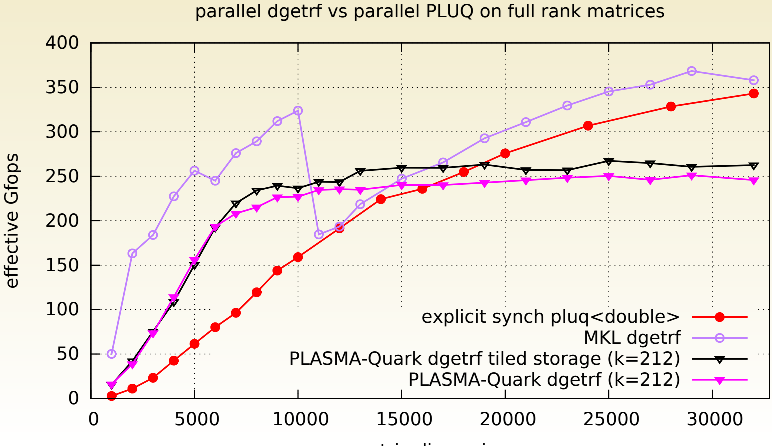


Puzzle game (block permutations)
Tile rec : better data locality and more square blocks for M.M.

State of the art : exact vs numerical linear algebra

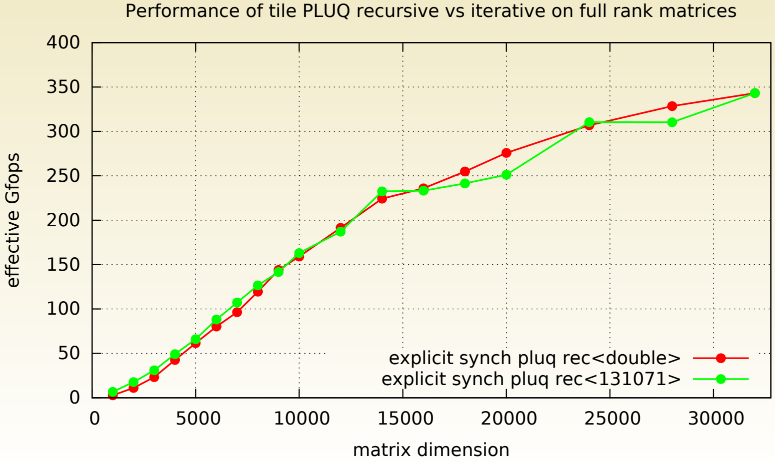
State of the art comparison :

- Exact PLUQ using PALADIn language : best performance with xKaapi
- Numerical LU (dgetrf) of PLASMA-Quark and MKL dgetrf



Performance of parallel PLUQ decomposition

Low impact of modular reductions in parallel
$\rightarrow$ Efficient SIMD implementation

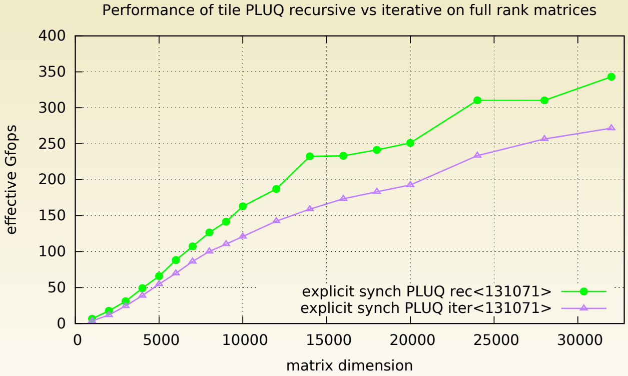


Modular reductions

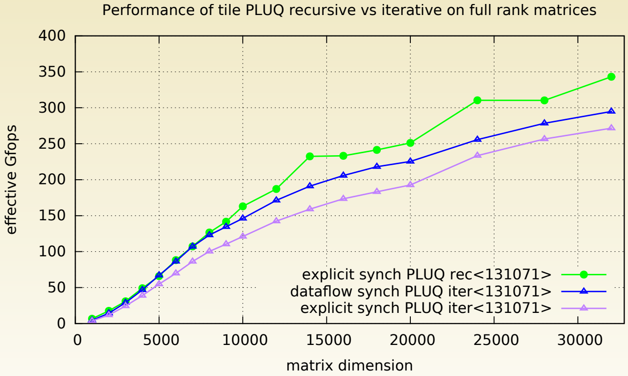
Iterative Right looking	$\mathbf{\frac{1}{3}n^3} - \frac{1}{3}n$
Iterative Left Looking	$\mathbf{\frac{3}{2}n^2} - \frac{5}{2}n + 1$
Iterative Crout	$\mathbf{\frac{3}{2}n^2} - \frac{5}{2}n + 1$
Tile Recursive	$\mathbf{2n^2} - n\log_2 n - 2n$
Slab Recursive	$(\mathbf{1} + \mathbf{\frac{1}{4}\log_2 n})\mathbf{n^2} - \frac{1}{2}n\log_2 n - n$

TABLE : Counting modular reductions in full rank LU factorization of an $n \times n$ matrix modulo p when $n(p-1)^2 < 2^{\text{mantissa}}$.

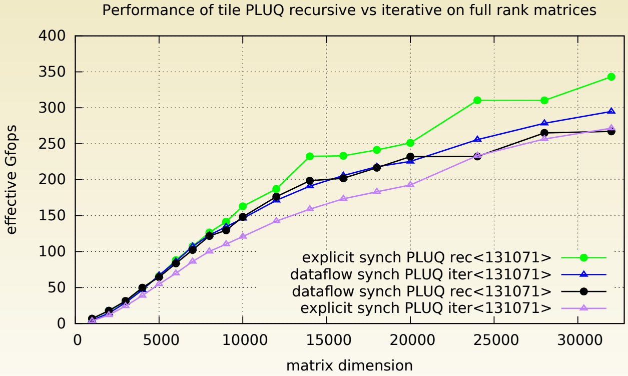
Performance of task parallelism : dataflow model



Performance of task parallelism : dataflow model



Performance of task parallelism : dataflow model



Possible improvement : implementation of the delegation of recursive tasks dependencies (Postpone access mode in the parallel programming environments)

Conclusion & Perspectives

HPAC DLP challenge : ~8 years → today feasible in ~3 months on 32 cores.

Defended theses

Sub-cubic : scale up in parallel in practice.

PALADIn : parallel programming environments as a plugin

The rank profile matrix : global information - efficient algorithms
Requires deep and precise understanding of pivoting

Perspectives

- Study the scaling of sub-cubic exact linear algebra algorithms on distributed machines.
- PALADIn on GPUs and distributed memory machines
- Adapt Communication avoiding algorithms to compute the rank profile information