Fast exact linear algebra: LinBox

Clément Pernet

SAGE Days 6,
November 11, 2007

Outline

LinBox: an overview
- Principles
- Organisation of the library
- Dense computations
- BlackBox computations

Memory efficient implementations
- In-place eliminations
- Fast matrix multiplication

Linear algebra over big integers

Outline

LinBox: an overview

- Principles
- Organisation of the library
- Dense computations
- BlackBox computations

Memory efficient implementations

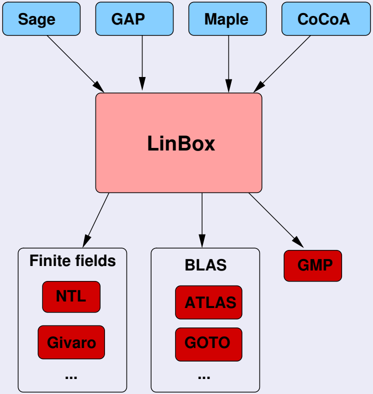
- In-place eliminations
- Fast matrix multiplication

Linear algebra over big integers

Outline

LinBox

A generic middleware



The LinBox project, facts

Joint NSF–CNRS project.

- U. of Delaware, North Carolina State U.
- U. of Waterloo, U. of Calgary,
- Laboratoires LJK, ID (Grenoble), LIP (Lyon)

The LinBox project, facts

Joint NSF–CNRS project.

- U. of Delaware, North Carolina State U.
- U. of Waterloo, U. of Calgary,
- Laboratoires LJK, ID (Grenoble), LIP (Lyon)

A LGPL source library:

- 122 000 lines of C++ code
- 5-10 active developers

Brief LinBox history

1998 Initial (and only) NSF-CNRS grant, first line of code. BlackBox linear algebra.

Aug 2002 v0.1 first beta release

Brief LinBox history

1998	Initial (and only) NSF-CNRS grant, first line of code. BlackBox linear algebra.
Aug 2002	v0.1 first beta release
March 2003	Introduction of `FFLAS-FFPACK` dense linear algebra kernel
July 2005	v1.0 release: first “user friendly” interface.

Brief LinBox history

1998	Initial (and only) NSF-CNRS grant, first line of code. BlackBox linear algebra.
Aug 2002	v0.1 first beta release
March 2003	Introduction of `FFLAS-FFPACK` dense linear algebra kernel
July 2005	v1.0 release: first "user friendly" interface.
Feb 2007	Wrapped in Sage

Brief LinBox history

1998 Initial (and only) NSF-CNRS grant, first line of code. BlackBox linear algebra.

Aug 2002 v0.1 first beta release

March 2003 Introduction of `FFLAS-FFPACK` dense linear algebra kernel

July 2005 v1.0 release: first “user friendly” interface.

Feb 2007 Wrapped in Sage

April 2007 creation of the public google-groups

Brief LinBox history

1998	Initial (and only) NSF-CNRS grant, first line of code. BlackBox linear algebra.
Aug 2002	v0.1 first beta release
March 2003	Introduction of `FFLAS-FFPACK` dense linear algebra kernel
July 2005	v1.0 release: first "user friendly" interface.
Feb 2007	Wrapped in Sage
April 2007	creation of the public google-groups
April 2007	first patch from M Abshoff
Oct 2007	v1.1.4

Brief LinBox history

1998	Initial (and only) NSF-CNRS grant, first line of code. BlackBox linear algebra.
Aug 2002	v0.1 first beta release
March 2003	Introduction of `FFLAS-FFPACK` dense linear algebra kernel
July 2005	v1.0 release: first "user friendly" interface.
Feb 2007	Wrapped in Sage
April 2007	creation of the public google-groups
April 2007	first patch from M Abshoff
Oct 2007	v1.1.4
2008	Towards a major change of interface (simpler) v2.0

LinBox-1.0

Solutions

- rank
- det
- minpoly
- charpoly
- system solve
- positive definiteness

LinBox-1.0

Solutions

- rank
- det
- minpoly
- charpoly
- system solve
- positive definiteness

Domains of computation

- Finite fields
- $\mathbb{Z}$

LinBox-1.0

Solutions

- rank
- det
- minpoly
- charpoly
- system solve
- positive definiteness

Domains of computation

- Finite fields
- $\mathbb{Z}$

Matrices

- Sparse, structured
- Dense

Outline

A design for genericity

Field/Ring interface

- Shared interface with Givaro
- Wraps NTL, Givaro implementations, using archetype or envelopes
- Proper implementations, suited for dense computations

A design for genericity

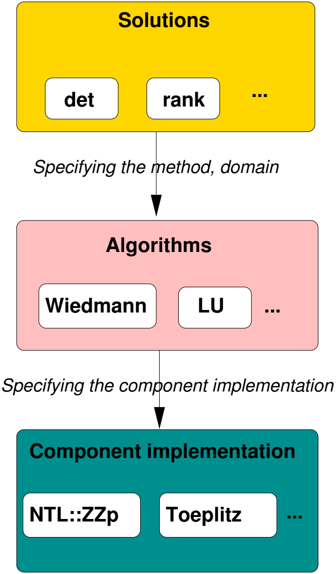
Field/Ring interface

- Shared interface with Givaro
- Wraps NTL, Givaro implementations, using archetype or envelopes
- Proper implementations, suited for dense computations

Matrix interface

- Sparse, Structured, Dense: BlackBox `apply`
- Dense matrix interface: several levels of abstraction

Structure of the library



Several levels of use

- **Web servers:** `http://www.linalg.org`
- **Executables:** `$ charpoly MyMatrix 65521`

Several levels of use

- Web servers: `http://www.linalg.org`
- Executables: `$ charpoly MyMatrix 65521`
- Call to a solution:

```
NTL::ZZp F(65521);
Toeplitz<NTL::ZZp> A(F);
Polynomial<NTL::ZZp> P;
charpoly (P, A);
```

Several levels of use

- Web servers: `http://www.linalg.org`
- Executables: `$ charpoly MyMatrix 65521`
- Call to a solution:

```
NTL::ZZp F(65521);
Toeplitz<NTL::ZZp> A(F);
Polynomial<NTL::ZZp> P;
charpoly (P, A);
```

- Calls to specific algorithms

Several levels of use

- Web servers: `http://www.linalg.org`
- Executables: `$ charpoly MyMatrix 65521`
- Call to a solution:

```
NTL::ZZp F(65521);
Toeplitz<NTL::ZZp> A(F);
Polynomial<NTL::ZZp> P;
charpoly (P, A);
```

- Calls to specific algorithms
- Hack with components

Outline

LinBox: an overview

- Principles
- Organisation of the library
- Dense computations
- BlackBox computations

Memory efficient implementations

- In-place eliminations
- Fast matrix multiplication

Linear algebra over big integers

Dense computations: FFLAS−FFPACK

Building block:

matrix multiplication over word-size finite field

Principle:

- Delayed modular reduction
- Floating point arithmetic (fused-mac, SSE2, ...)

Dense computations: `FFLAS-FFPACK`

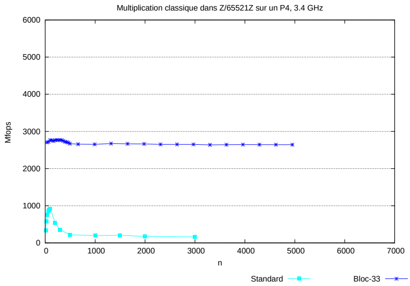
Building block:

matrix multiplication over word-size finite field

Principle:

- Delayed modular reduction
- Floating point arithmetic (fused-mac, SSE2, ...)
- cache tuning

$\Rightarrow$ rely on the existing BLAS



Dense computations: FFLAS–FFPACK

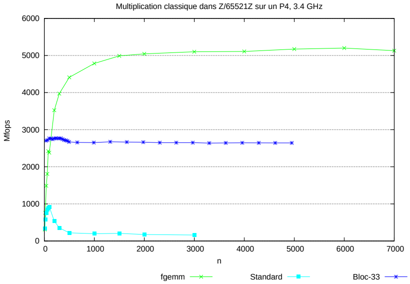
Building block:

matrix multiplication over word-size finite field

Principle:

- Delayed modular reduction
- Floating point arithmetic (fused-mac, SSE2, ...)
- cache tuning

⇒rely on the existing BLAS



Dense computations: FFLAS−FFPACK

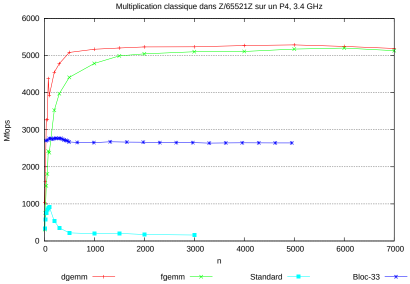
Building block:

matrix multiplication over word-size finite field

Principle:

- Delayed modular reduction
- Floating point arithmetic (fused-mac, SSE2, ...)
- cache tuning

⇒rely on the existing BLAS



Dense computations: FFLAS–FFPACK

Building block:

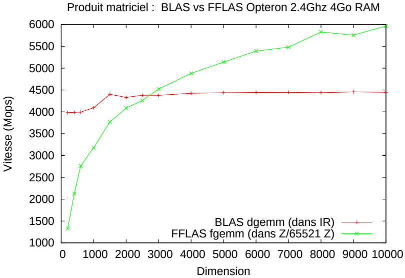
matrix multiplication over word-size finite field

Principle:

- Delayed modular reduction
- Floating point arithmetic (fused-mac, SSE2, ...)
- cache tuning

⇒rely on the existing BLAS

- Sub-cubic algorithm (Winograd)



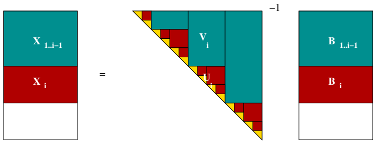
Design of other dense routines

- Reduction to matrix multiplication
- Bounds for delayed modular reductions.

Design of other dense routines

- Reduction to matrix multiplication
- Bounds for delayed modular reductions.

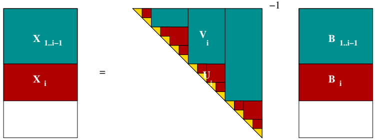
⇒Block algorithm with multiple cascade structures



Design of other dense routines

- Reduction to matrix multiplication
- Bounds for delayed modular reductions.

⇒Block algorithm with multiple cascade structures



	n	1000	2000	3000	5000	10 000
TRSM	$\frac{ftrsm}{dtrsm}$	1,66	1,33	1,24	1,12	1,01
LQUP	$\frac{lqup}{dgetrf}$	2,00	1,56	1,43	1,18	1,07
INVERSE	$\frac{inverse}{dgetrf+dgetri}$	1.62	1.32	1.15	0.86	0.76

Characteristic polynomial

Fact

$\mathcal{O}(n^\omega)$ Las Vegas probabilistic algorithm for the computation of the characteristic polynomial over a Field.

Characteristic polynomial

Fact

$\mathcal{O}(n^{\omega})$ Las Vegas probabilistic algorithm for the computation of the characteristic polynomial over a Field.

Practical algorithm :

Characteristic polynomial

Fact

$\mathcal{O}(n^\omega)$ Las Vegas probabilistic algorithm for the computation of the characteristic polynomial over a Field.

Practical algorithm :

n	500	5000	15 000
`LinBox`	0.91s	4m44s	2h20m
`magma-2.13`	1.27s	15m32s	7h28m

Characteristic polynomial

Fact

$\mathcal{O}(n^\omega)$ Las Vegas probabilistic algorithm for the computation of the characteristic polynomial over a Field.

Practical algorithm :

n	500	5000	15 000
`LinBox`	0.91s	4m44s	2h20m
`magma-2.13`	1.27s	15m32s	7h28m

- Frobenius normal form as well
- Transformation in $\mathcal{O}(n^\omega \log\log n)$

Outline

BlackBox computations

BlackBox computations

Goal: computation with very large sparse or structured matrices.

- No explicit representation of the matrix,
- Only operation: application of a vector

BlackBox computations

Goal: computation with very large sparse or structured matrices.

- No explicit representation of the matrix,
- Only operation: application of a vector
- Efficient algorithms
- Efficient preconditionners: Toeplitz, Hankel, Butterfly, ...

Block projection algorithms

- Wiedemann algorithm: scalar projections of A^i for $i = 1..2d$
- Block Wiedemann: $k \times k$ dense projections of A^i for $i = 1..2d/k$

$\Rightarrow$Balance efficiency between BlackBox and dense compations

Outline

Memory efficient dense linear algebra

Structure of dense algorithms: reduction to `matmul`

Memory efficient dense linear algebra

Structure of dense algorithms: reduction to `matmul`
Approach:

1. Memory efficient reductions to `matmul` (ideally in-place)
2. Reduce extra memory requirements for `matmul`

Outline

LinBox: an overview
- Principles
- Organisation of the library
- Dense computations
- BlackBox computations

Memory efficient implementations
- In-place eliminations
- Fast matrix multiplication

Linear algebra over big integers

Triangular decompositions

- Pre-Strassen, any rank profile:
  Turing, 48 : Gaussian elimination = LUP, in $\mathcal{O}\left(n^3\right)$

Triangular decompositions

- Pre-Strassen, any rank profile:
  Turing, 48 : Gaussian elimination = LUP, in $\mathcal{O}\left(n^3\right)$
- Post-Strassen, generic rank profile:
  Bunch, Hopcroft, 74 : $A = LU$, in $\mathcal{O}\left(n^\omega\right)$

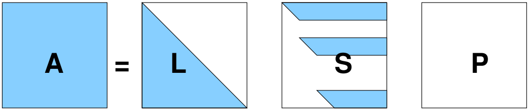
Triangular decompositions

- Pre-Strassen, any rank profile:
  Turing, 48 : Gaussian elimination = LUP, in $\mathcal{O}\left(n^3\right)$
- Post-Strassen, generic rank profile:
  Bunch, Hopcroft, 74 : $A = LU$, in $\mathcal{O}\left(n^\omega\right)$
- Post-Strassen, non singular:
  Bunch, Hopcroft, 74 : $A = LUP$, in $\mathcal{O}\left(n^\omega\right)$

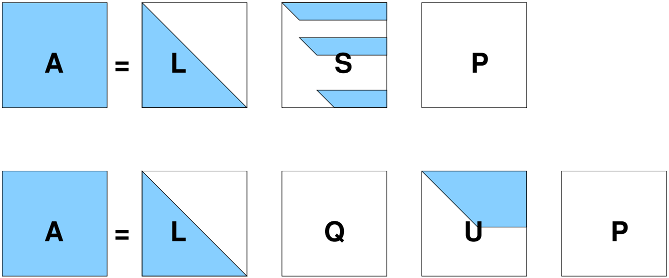
Triangular decompositions

- Pre-Strassen, any rank profile:
  Turing, 48 : Gaussian elimination = LUP, in $\mathcal{O}\left(n^3\right)$
- Post-Strassen, generic rank profile:
  Bunch, Hopcroft, 74 : $A = LU$, in $\mathcal{O}\left(n^\omega\right)$
- Post-Strassen, non singular:
  Bunch, Hopcroft, 74 : $A = LUP$, in $\mathcal{O}\left(n^\omega\right)$
- Post-Strassen any rank profile:
  Ibarra, Moran, Hui 82 : $A = LSP$, in $\mathcal{O}\left(n^\omega\right)$
  Ibarra, Moran, Hui 82 : $A = LQUP$, in $\mathcal{O}\left(n^\omega\right)$

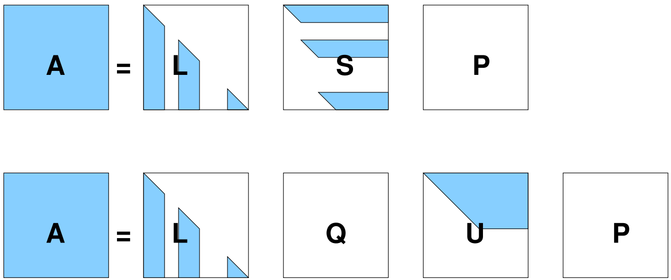
LSP-LQUP decompositions



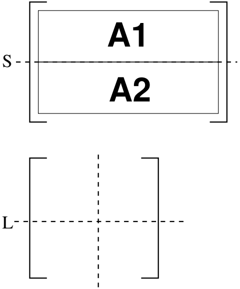
LSP-LQUP decompositions



LSP-LQUP decompositions

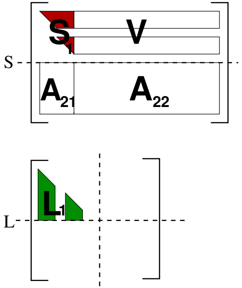


The LSP algorithm



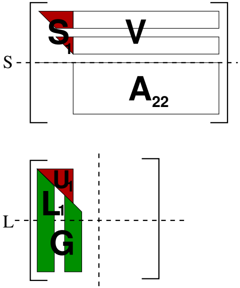
1. Split A Row-wise

The LSP algorithm



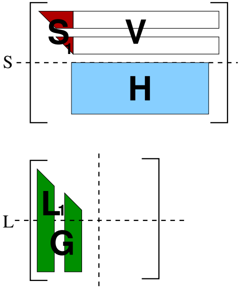
1. Split A Row-wise
2. Recursive call on A_1

The LSP algorithm



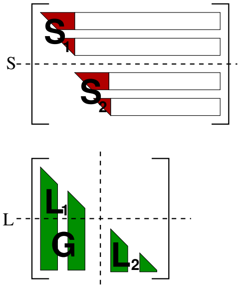
1. Split A Row-wise
2. Recursive call on A_1
3. $G \leftarrow A_{21} U_1^{-1}$ (trsm)

The LSP algorithm



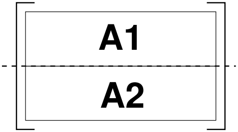
1. Split A Row-wise
2. Recursive call on A_1
3. $G \leftarrow A_{21} U_1^{-1}$ ($\texttt{trsm}$)
4. $H \leftarrow A_{22} - G \times V$ ($\texttt{matmul}$)

The LSP algorithm



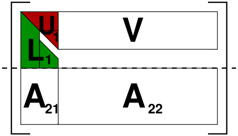
1. Split A Row-wise
2. Recursive call on A_1
3. $G \leftarrow A_{21} U_1^{-1}$ (`trsm`)
4. $H \leftarrow A_{22} - G \times V$ (`matmul`)
5. Recursive call on H

The LQUP decomposition



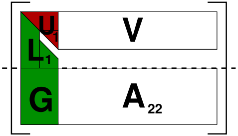
1. Split A Row-wise

The LQUP decomposition



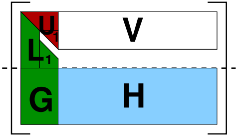
1. Split A Row-wise
2. Recursive call on A_1

The LQUP decomposition



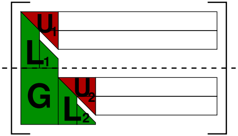
1. Split A Row-wise
2. Recursive call on A_1
3. $G \leftarrow A_{21} U_1^{-1}$ (`trsm`)

The LQUP decomposition



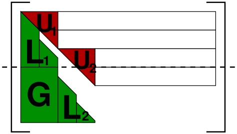
1. Split A Row-wise
2. Recursive call on A_1
3. $G \leftarrow A_{21} U_1^{-1}$ ($\mathtt{trsm}$)
4. $H \leftarrow A_{22} - G \times V$ ($\mathtt{matmul}$)

The LQUP decomposition



1. Split A Row-wise
2. Recursive call on A_1
3. $G \leftarrow A_{21} U_1^{-1}$ ($\texttt{trsm}$)
4. $H \leftarrow A_{22} - G \times V$ ($\texttt{matmul}$)
5. Recursive call on H

The LQUP decomposition



1. Split A Row-wise
2. Recursive call on A_1
3. $G \leftarrow A_{21} U_1^{-1}$ (`trsm`)
4. $H \leftarrow A_{22} - G \times V$ (`matmul`)
5. Recursive call on H
6. Row permutations

LSP-LQUP decompostions

Choice of the LQUP decomposition as a building block:

- in-place compact storage
- in-place computation

LSP-LQUP decompostions

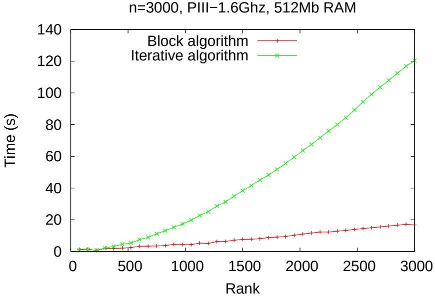
Choice of the LQUP decomposition as a building block:

- in-place compact storage
- in-place computation
- Permutation Q describes the row rank profile of A

LSP-LQUP decompostions

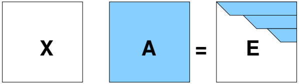
Choice of the LQUP decomposition as a building block:

- in-place compact storage
- in-place computation
- Permutation Q describes the row rank profile of A
- Rank sensitive computation time: $\mathcal{O}\left(mnr^{\omega-2}\right)$



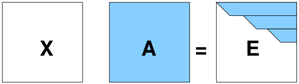
Echelon forms

Row Echelon Form $XA = E$

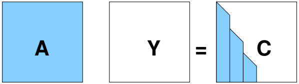


Echelon forms

Row Echelon Form $XA = E$

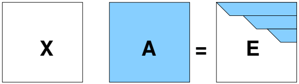


Column Echelon Form $AY = C$

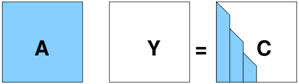


Echelon forms

Row Echelon Form $XA = E$



Column Echelon Form $AY = C$



Property (Link with LQUP)

$$C = LQ \begin{bmatrix} I_r \\ 0 \end{bmatrix}, \qquad Y = P^T \begin{bmatrix} U_1 & U_2 \\ 0 & I_{n-r} \end{bmatrix}^{-1}$$

From LQUP to Column Echelon

Additional operations:

- $-U^{-1}U_2$ `trsm` (triangular system solve) in-place

From LQUP to Column Echelon

Additional operations:

$-U^{-1}U_2$ `trsm` (triangular system solve) in-place

U_1^{-1}: `trtri` (triangular inverse)

From LQUP to Column Echelon

Additional operations:

$-U^{-1}U_2$ `trsm` (triangular system solve) in-place

U_1^{-1}: `trtri` (triangular inverse)

TRTRI: triangular inverse

$$\begin{bmatrix} U_1 & U_2 \\ & U_3 \end{bmatrix}^{-1} = \begin{bmatrix} U_1^{-1} & -U_1^{-1}U_2U_3^{-1} \\ & U_3^{-1} \end{bmatrix}$$

1: **if** $n = 1$ **then**
2: $\quad U \leftarrow U^{-1}$
3: **else**
4: $\quad U_2 \leftarrow U_3^{-1}U_2$ TRSM
5: $\quad U_2 \leftarrow -U_2U_3^{-1}$ TRSM
6: $\quad U_1 \leftarrow U_1^{-1}$ TRTRI
7: $\quad U_3 \leftarrow U_3^{-1}$ TRTRI
8: **end if**

From LQUP to Column Echelon

Additional operations:

$-U^{-1}U_2$ `trsm` (triangular system solve) in-place

U_1^{-1}: `trtri` (triangular inverse) in-place

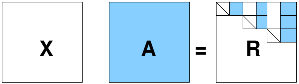
TRTRI: triangular inverse

$$\begin{bmatrix} U_1 & U_2 \\ & U_3 \end{bmatrix}^{-1} = \begin{bmatrix} U_1^{-1} & -U_1^{-1}U_2U_3^{-1} \\ & U_3^{-1} \end{bmatrix}$$

1: **if** $n = 1$ **then**
2: $\quad U \leftarrow U^{-1}$
3: **else**
4: $\quad U_2 \leftarrow U_3^{-1}U_2$ TRSM
5: $\quad U_2 \leftarrow -U_2U_3^{-1}$ TRSM
6: $\quad U_1 \leftarrow U_1^{-1}$ TRTRI
7: $\quad U_3 \leftarrow U_3^{-1}$ TRTRI
8: **end if**

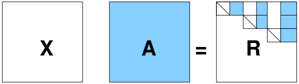
Reduced Echelon forms

Row Reduced Echelon Form $XA = E$

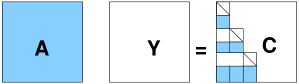


Reduced Echelon forms

Row Reduced Echelon Form $XA = E$



Column Reduced Echelon Form $AY = C$



From Echelon to Reduced Echelon

Again reduces to:

$U^{-1}X$: `TRSM`, in-place

U^{-1}: `TRTRI`, in-place

From Echelon to Reduced Echelon

Again reduces to:

$U^{-1}X$: `TRSM`, in-place

U^{-1}: `TRTRI`, in-place

UL: `TRTRM`,

From Echelon to Reduced Echelon

Again reduces to:

$U^{-1}X$: `TRSM`, in-place

U^{-1}: `TRTRI`, in-place

UL: `TRTRM`,

`TRTRM`: triangular triangular multiplication

$$\begin{bmatrix} U_1 & U_2 \\ & U_3 \end{bmatrix} \begin{bmatrix} L_1 & \\ L_2 & L_3 \end{bmatrix} = \begin{bmatrix} U_1L_1 + U_2L_2 & U_2L_3 \\ U_3L_2 & U_3L_3 \end{bmatrix}$$

1: $X_1 \leftarrow U_1L_1$ `TRTRM`
2: $X_1 \leftarrow X1 + U_2L_2$ `MM`
3: $X_2 \leftarrow U_2L_3$ `TRMM`
4: $X_3 \leftarrow U_3L_2$ `TRMM`
5: $X_4 \leftarrow U_3L_3$ `TRTRM`

From Echelon to Reduced Echelon

Again reduces to:

$U^{-1}X$: TRSM, in-place

U^{-1}: TRTRI, in-place

UL: TRTRM, in-place

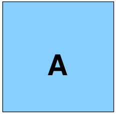
TRTRM: triangular triangular multiplication

$$\begin{bmatrix} U_1 & U_2 \\ & U_3 \end{bmatrix} \begin{bmatrix} L_1 & \\ L_2 & L_3 \end{bmatrix} = \begin{bmatrix} U_1 L_1 + U_2 L_2 & U_2 L_3 \\ U_3 L_2 & U_3 L_3 \end{bmatrix}$$

1: $X_1 \leftarrow U_1 L_1$ TRTRM
2: $X_1 \leftarrow X1 + U_2 L_2$ MM
3: $X_2 \leftarrow U_2 L_3$ TRMM
4: $X_3 \leftarrow U_3 L_2$ TRMM
5: $X_4 \leftarrow U_3 L_3$ TRTRM

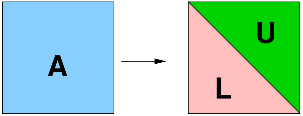
Example

A has full rank and generic rank profile.



Example

A has full rank and generic rank profile.

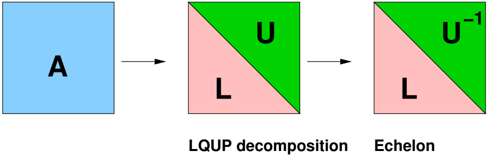


LQUP decomposition

$$A = LU$$

Example

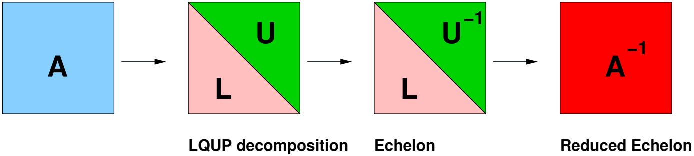
A has full rank and generic rank profile.



$$AU^{-1} = L$$

Example

A has full rank and generic rank profile.



$$A(U^{-1}L^{-1}) = I$$

Summary

Global scheme of reductions to LQUP decomposition.
Ensures:

- in-place computations

Summary

Global scheme of reductions to LQUP decomposition.
Ensures:

- in-place computations
- rank sensitive $\mathcal{O}\left(r^{\omega-2}n^2\right)$ computation time

Summary

Global scheme of reductions to LQUP decomposition.
Ensures:

- in-place computations
- rank sensitive $\mathcal{O}\left(r^{\omega-2}n^2\right)$ computation time
- increases modularity

Time complexity

These reductions are "efficient" with regard to the constant C_3 where $\mathcal{O}(n^3) = C_3 n^3$:

Time complexity

These reductions are "efficient" with regard to the constant C_3 where $\mathcal{O}(n^3) = C_3 n^3$:

	L, U, P	L, S, P	L, Q, U, P	Echelon	Red Echelon
cost	$2/3$	$2/3$	$2/3$	1	2

Time complexity

These reductions are “efficient” with regard to the constant C_3 where $\mathcal{O}\left(n^3\right) = C_3 n^3$:

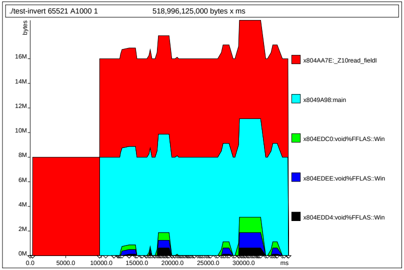
	L, U, P	L, S, P	L, Q, U, P	Echelon	Red Echelon
cost	$2/3$	$2/3$	$2/3$	1	2
rank profile	X	$2/3$	$2/3$	1	2
Echelon form	X	1	1	1	X
Red Echelon	X	2	2	2	2
in place	V	X	V	V	X

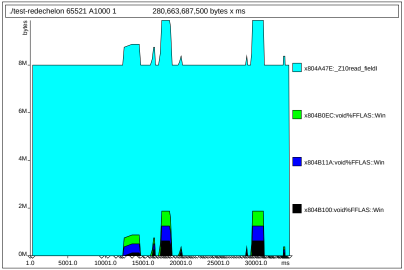
Time complexity

These reductions are “efficient” with regard to the constant C_3 where $\mathcal{O}\left(n^3\right) = C_3 n^3$:

	L, U, P	L, S, P	L, Q, U, P	Echelon	Red Echelon
cost	$2/3$	$2/3$	$2/3$	1	2
rank profile	X	$2/3$	$2/3$	1	2
Echelon form	X	1	1	1	X
Red Echelon	X	2	2	2	2
in place	V	X	V	V	X

Experiments





Outline

Strassen-Winograd algorithm

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix},$$

- 8 additions:

$$\begin{array}{ll} S_1 \leftarrow A_{21} + A_{22} & T_1 \leftarrow B_{12} - B_{11} \\ S_2 \leftarrow S_1 - A_{11} & T_2 \leftarrow B_{22} - T_1 \\ S_3 \leftarrow A_{11} - A_{21} & T_3 \leftarrow B_{22} - B_{12} \\ S_4 \leftarrow A_{12} - S_2 & T_4 \leftarrow T_2 - B_{21} \end{array}$$

- 7 recursive multiplications:

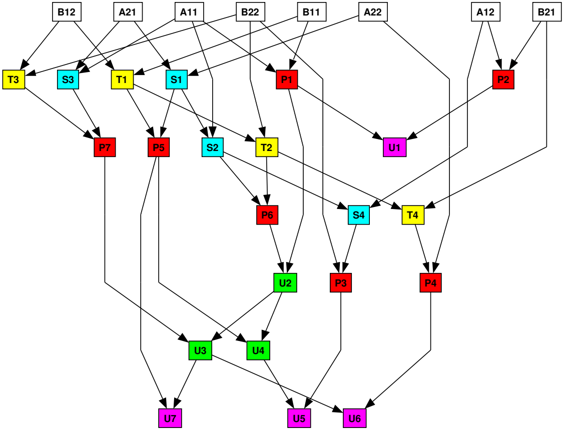
$$\begin{array}{ll} P_1 \leftarrow A_{11} \times B_{11} & P_5 \leftarrow S_1 \times T_1 \\ P_2 \leftarrow A_{12} \times B_{21} & P_6 \leftarrow S_2 \times T_2 \\ P_3 \leftarrow S_4 \times B_{22} & P_7 \leftarrow S_3 \times T_3 \\ P_4 \leftarrow A_{22} \times T_4 & \end{array}$$

- 7 final additions:

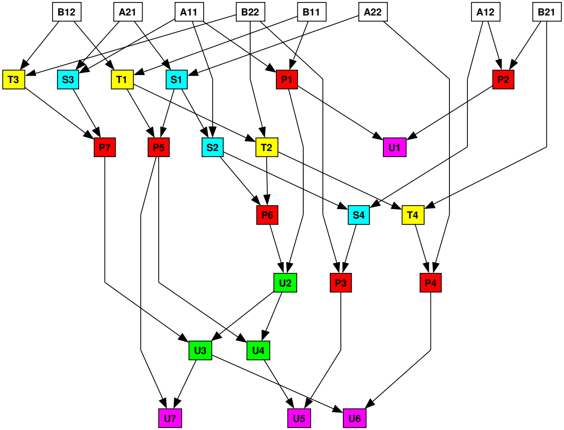
$$\begin{array}{ll} U_1 \leftarrow P_1 + P_2 & U_5 \leftarrow U_4 + P_3 \\ U_2 \leftarrow P_1 + P_6 & U_6 \leftarrow U_3 - P_4 \\ U_3 \leftarrow U_2 + P_7 & U_7 \leftarrow U_3 + P_5 \\ U_4 \leftarrow U_2 + P_5 & \end{array}$$

- The result is the matrix: $C = \begin{bmatrix} U1 & U5 \\ U6 & U7 \end{bmatrix}$

Tasks dependencies



Tasks dependencies



- Extra temporary blocks required
- Pebble game to minimize their number [Huss-Ledermann & Al. 96]

Reducing memory requirements

Dealing with 2 kind of computations:

- $C \leftarrow A \times B$
- $C \leftarrow A \times B + C$

Reducing memory requirements

Dealing with 2 kind of computations:

- $C \leftarrow A \times B$ — 2 temporaries $\Rightarrow 2/3n^2$
- $C \leftarrow A \times B + C$ — 3 temporaries $\Rightarrow n^2$

Previous work: [Huss-Ledermann & Al. 96].

Reducing memory requirements

Dealing with 2 kind of computations:

- $C \leftarrow A \times B$ 2 temporaries $\Rightarrow 2/3n^2$
- $C \leftarrow A \times B + C$ 3 temporaries $\Rightarrow n^2$

Previous work: [Huss-Ledermann & Al. 96].

Approach: relax some conditions

Reducing memory requirements

Dealing with 2 kind of computations:

- $C \leftarrow A \times B$ — 2 temporaries $\Rightarrow 2/3n^2$
- $C \leftarrow A \times B + C$ — 3 temporaries $\Rightarrow n^2$

Previous work: [Huss-Ledermann & Al. 96].

Approach: relax some conditions

- Inputs can be overwritten — $Cn^{2.8} + \epsilon n^{2.8}$

Reducing memory requirements

Dealing with 2 kind of computations:

- $C \leftarrow A \times B$ — 2 temporaries $\Rightarrow 2/3n^2$
- $C \leftarrow A \times B + C$ — 3 temporaries $\Rightarrow n^2$

Previous work: [Huss-Ledermann & Al. 96].

Approach: relax some conditions

- Inputs can be overwritten — $Cn^{2.8} + \epsilon n^{2.8}$
- Add a few pre-additions — $Cn^{2.8} + \epsilon n^{2.8}$

Reducing memory requirements

Dealing with 2 kind of computations:

- $C \leftarrow A \times B$ — 2 temporaries $\Rightarrow 2/3n^2$
- $C \leftarrow A \times B + C$ — 3 temporaries $\Rightarrow n^2$

Previous work: [Huss-Ledermann & Al. 96].

Approach: relax some conditions

- Inputs can be overwritten — $Cn^{2.8} + \epsilon n^{2.8}$
- Add a few pre-additions — $Cn^{2.8} + \epsilon n^{2.8}$
- Cascading with classical algorithm — $Cn^{2.8} + \epsilon n^{2.8}$

Results

Adding pre-additions:

#	operation	loc.	#ř	operation	loc.
1	$C_{22} = C_{22} - C_{12}$	C_{22}	13	$P_3 = S_4 B_{22} + C_{12}$	C_{12}
2	$C_{21} = C_{21} - C_{22}$	C_{21}	14	$P_1 = A_{11} B_{11}$	X_1
3	$C_{12} = C_{12} - C_{22}$	C_{12}	15	$U_2 = P_6 + P1$	C_{21}
4	$S_1 = A_{21} + A_{22}$	X_1	16	$P_2 = A_{12} B_{21} + C_{11}$	C_{11}
5	$T_1 = B_{12} - B_{11}$	X_2	17	$U_1 = P_1 + P_2$	C_{11}
6	$P_5 = S_1 T_1 + C_{12}$	C_{12}	18	$U_5 = U_2 + C_{12}$	C_{12}
7	$S_2 = S_1 - A_{11}$	X_1	19	$S_3 = A_{11} - A_{21}$	X_1
8	$T_2 = B_{22} - T_1$	X_2	20	$T_3 = B_{22} - B_{12}$	X_2
9	$P_6 = S_2 T_2 + C_{21}$	C_{21}	21	$U_3 = P_7 + U_2 = S_3 T_3 + U_2$	C_{21}
10	$S_4 = A_{12} - S_2$	X_1	22	$U_7 = U_3 + C_{22}$	C_{22}
11	$T_4 = T_2 - B_{21}$	X_2	23	$U_6 = U_3 - P_4 = -A_{12} T_4 + U_3$	C_{21}
12	$C_{22} = P_5 + C_{22}$	C_{22}			

- $C \leftarrow A \times B + C$ $\Rightarrow$ from 3 to 2 temp. (3 pre-adds)

Results

Overwriting inputs:

#	operation	loc.	#ř	operation	loc.
1	$C_{21} = C_{21} - C_{22}$	C_{21}	13	$P_4 = A_{22}T_4 + \beta C_{21}$	C_{21}
2	$C_{22} = C_{22} - C_{12}$	C_{22}	14	$P_2 = A_{12}B_{21} + \beta C_{11}$	C_{11}
3	$S_3 = A_{11} - A_{21}$	X	15	$P_1 = A_{11}B_{11}$	B_{21}
4	$T_3 = B_{22} - B_{12}$	Y	16	$U_1 = P_1 + P_2$	C_{11}
5	$P_7 = S_3T_3 + \beta C_{22}$	C_{22}	17	$P_6 = S_2T_2$	A_{12}
6	$S_1 = A_{21} + A_{22}$	A_{21}	18	$U_2 = P_1 + P_6$	C_{12}
7	$T_1 = B_{12} - B_{11}$	B_{12}	19	$U_4 = U_2 + P_5$	C_{12}
8	$S_2 = S_1 - A_{11}$	X	20	$U_3 = U_2 + P_7$	C_{22}
9	$T_2 = B_{22} - T_1$	Y	21	$U_7 = U_3 + P_5$	C_{22}
10	$P_5 = S_1T_1 + \beta C_{12}$	C_{12}	22	$U_6 = U_3 - P_4$	C_{21}
11	$S_4 = A_{12} - S_2$	A_{21}	23	$P_3 = S_4B_{22}$	A_{12}
12	$T_4 = T_2 - B_{21}$	B_{12}	24	$U_5 = U_4 + P_3$	C_{12}

- $C \leftarrow A \times B + C$ $\Rightarrow$from 3 to 2 temp. (2 pre-adds)

Results

Overwriting inputs:

#	operation	loc.	#ř	operation	loc.
1	$S_3 = A_{11} - A_{21}$	C_{11}	12	$S_4 = A_{12} - S_2$	C_{22}
2	$S_1 = A_{21} + A_{22}$	A_{21}	13	$P_6 = S_2 T_2$	C_{12}
3	$T_1 = B_{12} - B_{11}$	C_{22}	14	$U_2 = P_1 + P_6$	C_{12}
4	$T_3 = B_{22} - B_{12}$	B_{12}	15	$U_3 = U_2 + P_7$	C_{21}
5	$P_7 = S_3 T_3$	C_{21}	16	$P_3 = S_4 B_{22}$	B_{11}
6	$S_2 = S_1 - A_{11}$	B_{12}	17	$U_7 = U_3 + P_5$	C_{22}
7	$P_1 = A_{11} B_{11}$	C_{11}	18	$U_6 = U_3 - P_4$	C_{21}
8	$T_2 = B_{22} - T_1$	B_{11}	19	$U_4 = U_2 + P_5$	C_{12}
9	$P_5 = S_1 T_1$	A_{11}	20	$U_5 = U_4 + P_3$	C_{12}
10	$T_4 = T_2 - B_{21}$	C_{22}	21	$P_2 = A_{12} B_{21}$	B_{11}
11	$P_4 = A_{22} T_4$	A_{21}	22	$U_1 = P_1 + P_2$	C_{11}

- $C \leftarrow A \times B$ $\Rightarrow$fully in-place

Results

Overwriting inputs:

#	operation	loc.	#ř	operation	loc.
1	$S_3 = A_{11} - A_{21}$	C_{11}	12	$S_4 = A_{12} - S_2$	C_{22}
2	$S_1 = A_{21} + A_{22}$	A_{21}	13	$P_6 = S_2 T_2$	C_{12}
3	$T_1 = B_{12} - B_{11}$	C_{22}	14	$U_2 = P_1 + P_6$	C_{12}
4	$T_3 = B_{22} - B_{12}$	B_{12}	15	$U_3 = U_2 + P_7$	C_{21}
5	$P_7 = S_3 T_3$	C_{21}	16	$P_3 = S_4 B_{22}$	B_{11}
6	$S_2 = S_1 - A_{11}$	B_{12}	17	$U_7 = U_3 + P_5$	C_{22}
7	$P_1 = A_{11} B_{11}$	C_{11}	18	$U_6 = U_3 - P_4$	C_{21}
8	$T_2 = B_{22} - T_1$	B_{11}	19	$U_4 = U_2 + P_5$	C_{12}
9	$P_5 = S_1 T_1$	A_{11}	20	$U_5 = U_4 + P_3$	C_{12}
10	$T_4 = T_2 - B_{21}$	C_{22}	21	$P_2 = A_{12} B_{21}$	B_{11}
11	$P_4 = A_{22} T_4$	A_{21}	22	$U_1 = P_1 + P_2$	C_{11}

- $C \leftarrow A \times B$ $\Rightarrow$fully in-place

Question:

Is there an in-place $\mathcal{O}\left(n^{2.807}\right)$ algorithm with constant inputs?

Results

Overwriting inputs:

#	operation	loc.	#ř	operation	loc.
1	$S_3 = A_{11} - A_{21}$	C_{11}	12	$S_4 = A_{12} - S_2$	C_{22}
2	$S_1 = A_{21} + A_{22}$	A_{21}	13	$P_6 = S_2 T_2$	C_{12}
3	$T_1 = B_{12} - B_{11}$	C_{22}	14	$U_2 = P_1 + P_6$	C_{12}
4	$T_3 = B_{22} - B_{12}$	B_{12}	15	$U_3 = U_2 + P_7$	C_{21}
5	$P_7 = S_3 T_3$	C_{21}	16	$P_3 = S_4 B_{22}$	B_{11}
6	$S_2 = S_1 - A_{11}$	B_{12}	17	$U_7 = U_3 + P_5$	C_{22}
7	$P_1 = A_{11} B_{11}$	C_{11}	18	$U_6 = U_3 - P_4$	C_{21}
8	$T_2 = B_{22} - T_1$	B_{11}	19	$U_4 = U_2 + P_5$	C_{12}
9	$P_5 = S_1 T_1$	A_{11}	20	$U_5 = U_4 + P_3$	C_{12}
10	$T_4 = T_2 - B_{21}$	C_{22}	21	$P_2 = A_{12} B_{21}$	B_{11}
11	$P_4 = A_{22} T_4$	A_{21}	22	$U_1 = P_1 + P_2$	C_{11}

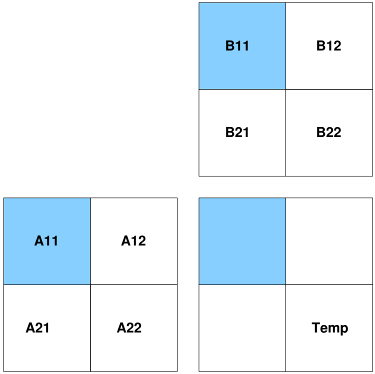
- $C \leftarrow A \times B$ $\Rightarrow$fully in-place

Question:

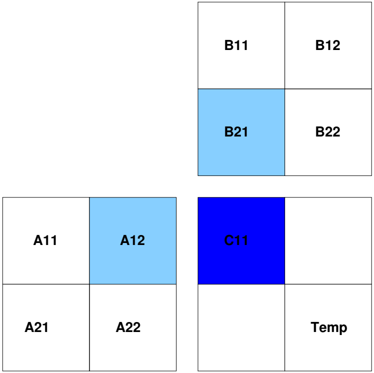
Is there an in-place $\mathcal{O}\left(n^{2.807}\right)$ algorithm with constant inputs?

$\Rightarrow$yes

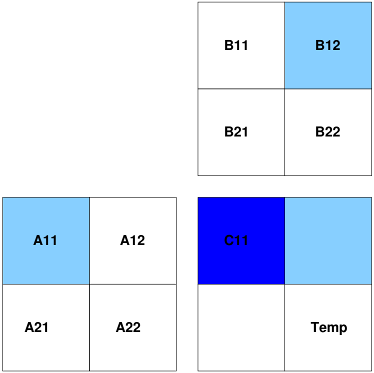
Principle of the fully in-place algorithm



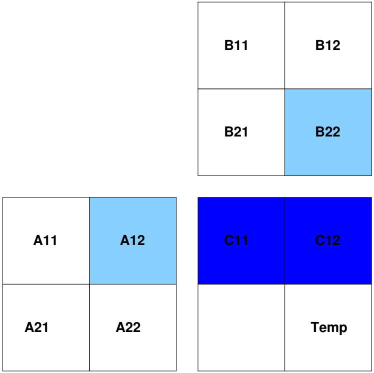
Principle of the fully in-place algorithm



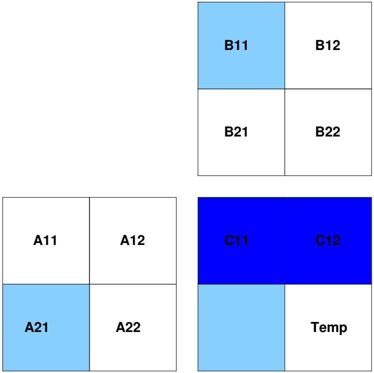
Principle of the fully in-place algorithm



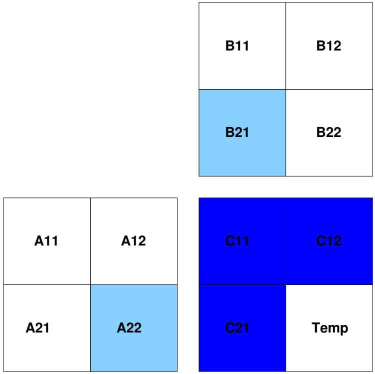
Principle of the fully in-place algorithm



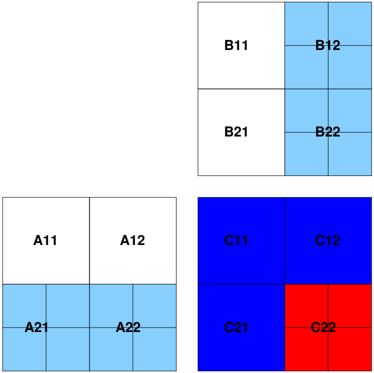
Principle of the fully in-place algorithm



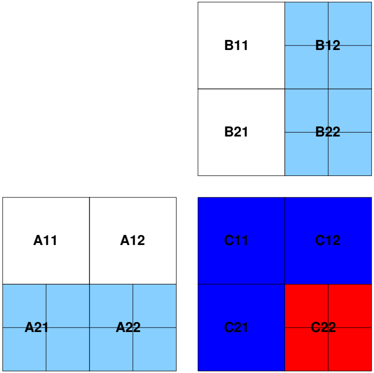
Principle of the fully in-place algorithm



Principle of the fully in-place algorithm



Principle of the fully in-place algorithm



- $7.2n^{2.807}$ instead of $6n^{2.807}$

Outline

LinBox: an overview

- Principles
- Organisation of the library
- Dense computations
- BlackBox computations

Memory efficient implementations

- In-place eliminations
- Fast matrix multiplication

Linear algebra over big integers

The problem

- Reasonably small dimension ($n = 2..100$)
- Unreasonably large entries
  ($\log_2 \|A\|_\infty = 1,000,000..1,000,000,000$)

The problem

- Reasonably small dimension ($n = 2..100$)
- Unreasonably large entries ($\log_2 \|A\|_\infty = 1,000,000..1,000,000,000$)

$$\texttt{mul} \gg \texttt{add}$$

despite FFT

The problem

- Reasonably small dimension ($n = 2..100$)
- Unreasonably large entries ($\log_2 \|A\|_\infty = 1,000,000..1,000,000,000$)

$$\texttt{mul} \gg \texttt{add}$$

despite FFT

- Fast Matrix Multiplication is always better than classic,
- Can do better than Strassen-Winograd

Dealing with odd dimensions

Padding: add 0 columns and rows to the nearest power of 2 (more operations)

Peeling: slice down to the nearest power of 2, and use classical block algorithm (less “sub-cubic”).

Dealing with odd dimensions

Padding: add 0 columns and rows to the nearest power of 2 (more operations)

Peeling: slice down to the nearest power of 2, and use classical block algorithm (less “sub-cubic”).

Static: Before actual computation.

Dynamic: At each recursive level, dimension 1 modifications.

Dealing with odd dimensions

Padding: add 0 columns and rows to the nearest power of 2 (more operations)

Peeling: slice down to the nearest power of 2, and use classical block algorithm (less "sub-cubic").

Static: Before actual computation.

Dynamic: At each recursive level, dimension 1 modifications.

Virtual dynamic padding:

- Recursive splitting with odd dimensions
- No extra operations (virtual 0)
- Better operation count than peeling

Winograd 68

Formula for dot-product:

$$a_1 b_1 + a_2 b_2 = (a_1 + b_2)(a_2 + b_1) - a_1 a_2 - b_1 b_2$$

Winograd 68

Formula for dot-product:

$$a_1 b_1 + a_2 b_2 = (a_1 + b_2)(a_2 + b_1) - a_1 a_2 - b_1 b_2$$

for k=1..n/2 **do**
  for i=1..n **do**
    $\alpha_{i,k} = (a_{i,2k} a_{i,2k+1})$
  end for
  for j=1..n **do**
    $\beta_{k,j} = (b_{2k,j} b_{2k+1,j})$
  end for
  for i=1..n **do**
    for j=1..n **do**
      $C_{i,j} += (a_{i,2k} + b_{2k+1,j})(a_{i,2k+1} + b_{2k,j}) - \alpha_{i,k} - \beta_{k,j}$
    end for
  end for
end for

Winograd 68

Formula for dot-product:

$$a_1 b_1 + a_2 b_2 = (a_1 + b_2)(a_2 + b_1) - a_1 a_2 - b_1 b_2$$

for k=1..n/2 **do**
    for i=1..n **do**
        $\alpha_{i,k} = (a_{i,2k} a_{i,2k+1})$
    end for
    for j=1..n **do**
        $\beta_{k,j} = (b_{2k,j} b_{2k+1,j})$
    end for
    for i=1..n **do**
        for j=1..n **do**
            $C_{i,j} += (a_{i,2k} + b_{2k+1,j})(a_{i,2k+1} + b_{2k,j}) - \alpha_{i,k} - \beta_{k,j}$
        end for
    end for
end for

- Requires commutativity (no recursive algorithm)
- Still $\mathcal{O}(n^3)$
- But better constant: $T_2(n) = 1/2n^3 + n^2$ instead of $1n^3$

From 2 to 3

$$a_1 b_1 + a_2 b_2 + a_3 b_3 = (a_1 + a_2 + b_3)(a_3 + b_1 + b_2)$$

$$T_3(n) = 1/3n^3$$

From 2 to 3

$$a_1b_1 + a_2b_2 + a_3b_3 = (a_1 + a_2 + b_3)(a_3 + b_1 + b_2) - (a_1 + a_2)a_3 - b_3(b_1 + b_2)$$

$$T_3(n) = 1/3n^3 \qquad + 2/3n^2$$

From 2 to 3

$$\begin{aligned} a_1 b_1 + a_2 b_2 + a_3 b_3 &= (a_1 + a_2 + b_3)(a_3 + b_1 + b_2) \\ &\quad - (a_1 + a_2) a_3 - b_3 (b_1 + b_2) \\ &\quad - a_1 b_2 - a_2 b_1 \end{aligned}$$

$$T_3(n) = 1/3 n^3 + T(n, 2/3 n) + 2/3 n^2$$

Comparison

n	2	3	4	6	8
Classical algorithm	8	27	64	216	512
Strassen+Peeling	**7**	26	49	182	343

n	9	10	12	15	18
Classical algorithm	729	1000	1728	3375	5832
Strassen+Peeling	560	770	1274	2794	3920

Comparison

n	2	3	4	6	8
Classical algorithm	8	27	64	216	512
Strassen+Peeling	**7**	26	49	182	343
Strassen+virt. padding	**7**	25	49	175	343

n	9	10	12	15	18
Classical algorithm	729	1000	1728	3375	5832
Strassen+Peeling	560	770	1274	2794	3920
Strassen+virt. padding		763	1225	2161	3401

Comparison

n	2	3	4	6	8
Classical algorithm	8	27	64	216	512
Strassen+Peeling	**7**	26	49	182	343
Strassen+virt. padding	**7**	25	49	175	343
Winograd 68	8		**48**	**144**	**320**

n	9	10	12	15	18
Classical algorithm	729	1000	1728	3375	5832
Strassen+Peeling	560	770	1274	2794	3920
Strassen+virt. padding	559	763	1225	2161	3401
Winograd 68		**600**	**1008**		**3240**

Comparison

n	2	3	4	6	8
Classical algorithm	8	27	64	216	512
Strassen+Peeling	**7**	26	49	182	343
Strassen+virt. padding	**7**	25	49	175	343
Winograd 68	8		**48**	**144**	**320**
New algorithm		**24**		158	

n	9	10	12	15	18
Classical algorithm	729	1000	1728	3375	5832
Strassen+Peeling	560	770	1274	2794	3920
Strassen+virt. padding	559	763	1225	2161	3401
Winograd 68		**600**	**1008**		**3240**
New algorithm	**489**		1088	**2093**	3456

Perspectives

- Study extensively most small case algorithm,
- ...including rectangular matrices,
- ...including [Bini, Cappovani & Al.] $\mathcal{O}\left(n^{2.779}\right)$

Perspectives

- Study extensively most small case algorithm,
- ...including rectangular matrices,
- ...including [Bini, Cappovani & Al.] $\mathcal{O}\left(n^{2.779}\right)$
- build a database for small dimensions,

Perspectives

- Study extensively most small case algorithm,
- ...including rectangular matrices,
- ...including [Bini, Cappovani & Al.] $\mathcal{O}\left(n^{2.779}\right)$
- build a database for small dimensions,
- automatically generate a combination of base case algorithms for a given dimension