Solving Quadratic Equations with XL on Parallel Architectures

Cheng Chen-Mou[1], Chou Tung[2],
Ni Ru-Ben[2], Yang Bo-Yin[2]

[1]National Taiwan University
[2]Academia Sinica
Taipei, Taiwan

Leuven, Sept. 11, 2012

Solving Quadratic Equations with XL on Parallel Architectures

Chen-Mou Cheng[1], Tung Chou[2],
Ruben Niederhagen[2], Bo-Yin Yang[2]

[1]National Taiwan University
[2]Academia Sinica
Taipei, Taiwan

Leuven, Sept. 11, 2012

The XL algorithm

Some cryptographic systems can be attacked by solving a system of multivariate quadratic equations, e.g.:

- AES:
  - 8000 quadratic equations with 1600 variables over $\mathbb{F}_2$ (Courtois and Pieprzyk, 2002)
  - 840 sparse quadratic equations and 1408 linear equations over 3968 variables of $\mathbb{F}_{256}$ (Murphy and Robshaw, 2002)
- multivariate cryptographic systems, e.g. QUAD stream cipher (cryptanalysis by Yang, Chen, Bernstein, and Chen, 2007)

The XL algorithm

- *XL* is an acronym for *extended linearization*:
  - *extend* a quadratic system by multiplying with appropriate monomials
  - *linearize* by treating each monomial as an independent variable
  - solve the linearized system
- special case of Gröbner basis algorithms
- first suggested by Lazard (1983)
- reinvented by Courtois, Klimov, Patarin, and Shamir (2000)
- alternative to Gröbner basis solvers like Faugère's F_4 (1999, e.g., Magma) and F_5 (2002) algorithms

The XL algorithm

For $b \in \mathbb{N}^n$ denote by x^b the monomial $x_1^{b_1} x_2^{b_2} \dots x_n^{b_n}$ and by $|b| = b_1 + b_2 + \dots + b_n$ the total degree of x^b.

given:	finite field $K = \mathbb{F}_q$
	system $\mathcal{A}$ of m multivariate quadratic equations:
	$\ell_1 = \ell_2 = \dots = \ell_m = 0,\ \ell_i \in K[x_1, x_2, \dots, x_n]$
choose:	operational degree $D \in \mathbb{N}$
extend:	system $\mathcal{A}$ to the system
	$\mathcal{R}^{(D)} = \{x^b \ell_i = 0 : \lvert b \rvert \leq D - 2, \ell_i \in \mathcal{A}\}$
linearize:	consider $x^d, d \leq D$ a new variable
	to obtain a linear system $\mathcal{M}$
solve:	linear system $\mathcal{M}$

The XL algorithm

For $b \in \mathbb{N}^n$ denote by x^b the monomial $x_1^{b_1} x_2^{b_2} \dots x_n^{b_n}$ and by $|b| = b_1 + b_2 + \dots + b_n$ the total degree of x^b.

given:	finite field $K = \mathbb{F}_q$
	system $\mathcal{A}$ of m multivariate quadratic equations:
	$\ell_1 = \ell_2 = \dots = \ell_m = 0,\ \ell_i \in K[x_1, x_2, \dots, x_n]$
choose:	$\underline{\text{operational degree } D \in \mathbb{N}}$ How?
extend:	system $\mathcal{A}$ to the system
	$\mathcal{R}^{(D)} = \{x^b \ell_i = 0 : \lvert b \rvert \leq D - 2, \ell_i \in \mathcal{A}\}$
linearize:	consider $x^d, d \leq D$ a new variable
	to obtain a linear system $\mathcal{M}$
solve:	linear system $\mathcal{M}$

The XL algorithm

For $b \in \mathbb{N}^n$ denote by x^b the monomial $x_1^{b_1} x_2^{b_2} \dots x_n^{b_n}$ and by $|b| = b_1 + b_2 + \cdots + b_n$ the total degree of x^b.

given:	finite field $K = \mathbb{F}_q$
	system $\mathcal{A}$ of m multivariate quadratic equations:
	$\ell_1 = \ell_2 = \cdots = \ell_m = 0,\ \ell_i \in K[x_1, x_2, \dots, x_n]$
choose:	<u>operational degree $D \in \mathbb{N}$</u> How?
extend:	system $\mathcal{A}$ to the system
	$\mathcal{R}^{(D)} = \{x^b \ell_i = 0 : \lvert b\rvert \le D - 2, \ell_i \in \mathcal{A}\}$
linearize:	consider $x^d, d \le D$ a new variable
	to obtain a linear system $\mathcal{M}$
solve:	linear system $\mathcal{M}$

minimum degree D_0 for reliable termination (Yang and Chen):

$$D_0 := \min\{D : ((1-\lambda)^{m-n-1}(1+\lambda)^m)[D] \le 0\}$$

The XL algorithm

For $b \in \mathbb{N}^n$ denote by x^b the monomial $x_1^{b_1} x_2^{b_2} \ldots x_n^{b_n}$ and by $|b| = b_1 + b_2 + \cdots + b_n$ the total degree of x^b.

given:	finite field $K = \mathbb{F}_q$
	system $\mathcal{A}$ of m multivariate quadratic equations:
	$\ell_1 = \ell_2 = \cdots = \ell_m = 0,\ \ell_i \in K[x_1, x_2, \ldots, x_n]$
choose:	$\underline{\text{operational degree } D \in \mathbb{N}}$ How?
extend:	system $\mathcal{A}$ to the system
	$\mathcal{R}^{(D)} = \{x^b \ell_i = 0 : \lvert b\rvert \leq D - 2, \ell_i \in \mathcal{A}\}$
linearize:	consider $x^d, d \leq D$ a new variable
	to obtain a linear system $\mathcal{M}$
solve:	$\underline{\text{linear system } \mathcal{M}}$ How?

minimum degree D_0 for reliable termination (Yang and Chen):

$$D_0 := \min\{D : ((1-\lambda)^{m-n-1}(1+\lambda)^m)[D] \leq 0\}$$

The XL algorithm

For $b \in \mathbb{N}^n$ denote by x^b the monomial $x_1^{b_1} x_2^{b_2} \ldots x_n^{b_n}$ and by $|b| = b_1 + b_2 + \cdots + b_n$ the total degree of x^b.

given: finite field $K = \mathbb{F}_q$
system $\mathcal{A}$ of m multivariate quadratic equations:
$\ell_1 = \ell_2 = \cdots = \ell_m = 0$, $\ell_i \in K[x_1, x_2, \ldots, x_n]$

choose:

extend:

linearize:

to obtain a linear system $\mathcal{M}$

solve: linear system $\mathcal{M}$ How?

We use the Wiedemann algorithm instead of a Gauss solver. Thus, we do not compute a complete Gröbner basis but distinguished solutions!

minimum degree D_0 for reliable termination (Yang and Chen):

$$D_0 := \min\{D : ((1-\lambda)^{m-n-1}(1+\lambda)^m)[D] \leq 0\}$$

The Wiedemann algorithm – the basic idea

given: $A \in K^{N\times N}$
wanted: $v \in K^N$ such that $Av = 0$
solution: compute minimal polynomial f of A of degree d: $f(A) = 0$

$$\sum_{i=0}^{d} f_i A^i = 0$$

$$\sum_{i=0}^{d} f_i A^i z = 0 \quad \text{choose } z \in K^N \text{ randomly}$$

$$\sum_{i=1}^{d} f_i A^i z + f_0 z = 0$$

$$\sum_{i=1}^{d} f_i A^i z = 0 \quad \text{since } f_0 = 0$$

$$A \cdot \underbrace{\left(\sum_{i=1}^{d} f_i A^{i-1} z \right)}_{=v} = 0$$

The Wiedemann algorithm – the basic idea

given: $A \in K^{N \times N}$

wanted: $v \in K^N$ such that $Av = 0$ How?

solution: compute minimal polynomial f of A of degree d: $f(A) = 0$

$$\sum_{i=0}^{d} f_i A^i = 0$$

$$\sum_{i=0}^{d} f_i A^i z = 0 \quad \text{choose } z \in K^N \text{ randomly}$$

$$\sum_{i=1}^{d} f_i A^i z + f_0 z = 0$$

$$\sum_{i=1}^{d} f_i A^i z = 0 \quad \text{since } f_0 = 0$$

$$A \cdot \underbrace{\left(\sum_{i=1}^{d} f_i A^{i-1} z \right)}_{=v} = 0$$

The Berlekamp–Massey algorithm

Given a linearly recurrent sequence

$$S = \{a_i\}_{i=0}^{\infty},\ a_i \in K,$$

compute an *annihilating* polynomial f of degree d such that

$$\sum_{i=0}^{d} f_i a_{j+i} = 0, \text{ for all } j \in \mathbb{N}.$$

The Berlekamp–Massey algorithm requires the first $2 \cdot d$ elements of S as input and computes $f_i \in K,\ 0 \leq i \leq d$.

The Berlekamp–Massey algorithm

$a_i = xA^i z, \; x \in K^{1\times N}$

Given a linearly recurrent sequence

$$S = \{a_i\}_{i=0}^{\infty}, \; a_i \in K,$$

compute an *annihilating* polynomial f of degree d such that

$$\sum_{i=0}^{d} f_i a_{j+i} = 0, \text{ for all } j \in \mathbb{N}.$$

The Berlekamp–Massey algorithm requires the first $2 \cdot d$ elements of S as input and computes $f_i \in K, \; 0 \le i \le d$.

The block Wiedemann algorithm

Due to Coppersmith (1994), three steps:

Input: $A \in K^{N\times N}$, parameters $m, n \in \mathbb{N}$, $\kappa \in \mathbb{N}$ of size $N/m + N/n + O(1)$.

The block Wiedemann algorithm

Due to Coppersmith (1994), three steps:

Input: $A \in K^{N \times N}$, parameters $m, n \in \mathbb{N}$, $\kappa \in \mathbb{N}$ of size $N/m + N/n + O(1)$.

BW1: Compute sequence $\{a_i\}_{i=0}^{\kappa}$ of matrices $a_i \in K^{n \times m}$ using random matrices $x \in K^{m \times N}$ and $z \in K^{N \times n}$

$$a_i = (xA^i y)^T, \quad \text{for } y = Az. \quad \boxed{O\left(N^2(w_A + m)\right)}$$

The block Wiedemann algorithm

Due to Coppersmith (1994), three steps:

Input: $A \in K^{N\times N}$, parameters $m, n \in \mathbb{N}$, $\kappa \in \mathbb{N}$ of size $N/m + N/n + O(1)$.

BW1: Compute sequence $\{a_i\}_{i=0}^{\kappa}$ of matrices $a_i \in K^{n\times m}$ using random matrices $x \in K^{m\times N}$ and $z \in K^{N\times n}$

$$a_i = (xA^i y)^T, \quad \text{for } y = Az. \quad \boxed{O\left(N^2(w_A + m)\right)}$$

BW2: Use block Berlekamp–Massey to compute polynomial f with coefficients in $K^{n\times n}$.

Coppersmith's version: $\boxed{O(N^2 \cdot n)}$

Thomé's version: $\boxed{O(N \log^2 N \cdot n)}$

The block Wiedemann algorithm

Due to Coppersmith (1994), three steps:

Input: $A \in K^{N\times N}$, parameters $m, n \in \mathbb{N}$, $\kappa \in \mathbb{N}$ of size $N/m + N/n + O(1)$.

BW1: Compute sequence $\{a_i\}_{i=0}^{\kappa}$ of matrices $a_i \in K^{n\times m}$ using random matrices $x \in K^{m\times N}$ and $z \in K^{N\times n}$

$$a_i = (xA^i y)^T, \quad \text{for } y = Az. \quad \boxed{O\left(N^2(w_A + m)\right)}$$

BW2: Use block Berlekamp–Massey to compute polynomial f with coefficients in $K^{n\times n}$.

Coppersmith's version: $\boxed{O(N^2 \cdot n)}$

Thomé's version: $\boxed{O(N \log^2 N \cdot n)}$

BW3: Evaluate the reverse of f:

$$W = \sum_{j=0}^{\deg(f)} A^j z (f_{\deg(f)-j})^T. \quad \boxed{O\left(N^2(w_A + n)\right)}$$

Parallelization of BW1

Input: $A \in K^{N \times N}$, parameters $m, n \in \mathbb{N}$, $\kappa \in \mathbb{N}$ of size $N/m + N/n + O(1)$.

Compute sequence $\{a_i\}_{i=0}^{\kappa}$ of matrices $a_i \in K^{n \times m}$ using random matrices $x \in K^{m \times N}$ and $z \in K^{N \times n}$

$$a_i = (xA^i y)^T, \quad \text{for } y = Az$$

using $\{t_i\}_{i=0}^{\kappa}$, $t_i \in K^{N \times n}$,

$$t_i = \begin{cases} y = Az & \text{for } i = 0 \\ At_{i-1} & \text{for } 0 < i \leq \kappa, \end{cases}$$

$$a_i = (xt_i)^T.$$

Parallelization of BW1

```
INPUT: macaulay_matrix<N, N> A;
       sparse_matrix<N, n> z;
matrix<N, n> t_new, t_old;
matrix<m, n> a[N/m + N/n + O(1)];
sparse_matrix<m, N, weight> x;

x.rand();
t_old = z;
for (unsigned i = 0; i <= N/m + N/n + O(1); i++)
{
    t_new = A * t_old;
    a[i] = x * t_new;
    swap(t_old, t_new);
}

RETURN a
```

Parallelization of BW1 – multicore processor

$$t_i = A \times t_{i-1}$$

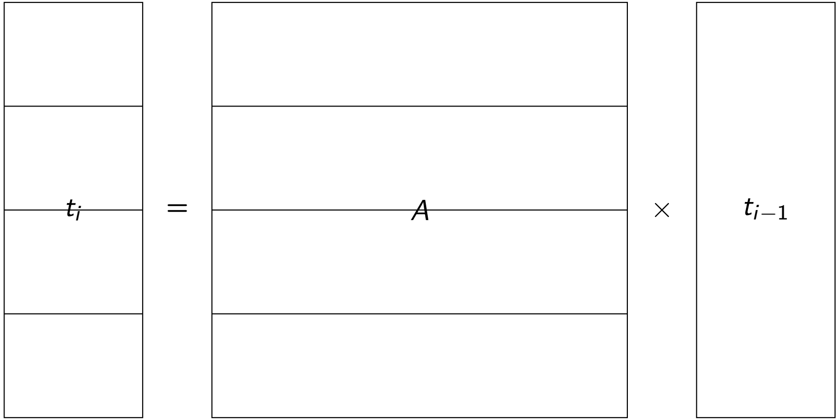
Parallelization of BW1 – multicore processor

2 cores

$$t_i = A \times t_{i-1}$$

Parallelization of BW1 – multicore processor

4 cores



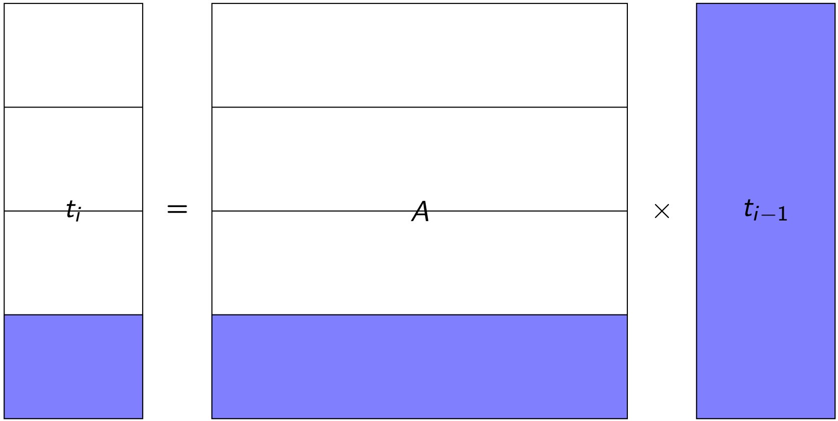
Parallelization of BW1 – multicore processor

4 cores

$$t_i = A \times t_{i-1}$$

Parallelization of BW1 – multicore processor

4 cores



OpenMP

Parallelization of BW1 – cluster system

$$t_i = A \times t_{i-1}$$

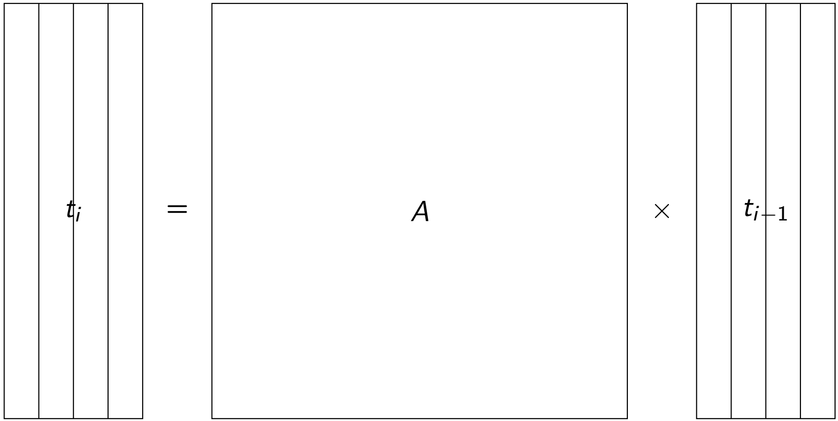
Parallelization of BW1 – cluster system

2 computing nodes

$$t_i = A \times t_{i-1}$$

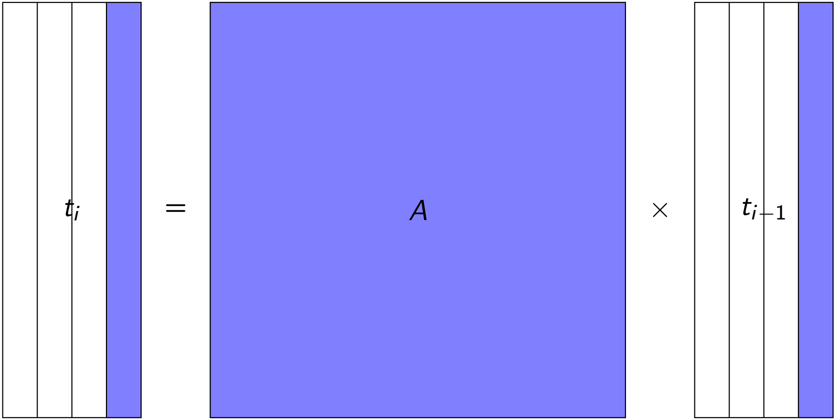
Parallelization of BW1 – cluster system

4 computing nodes



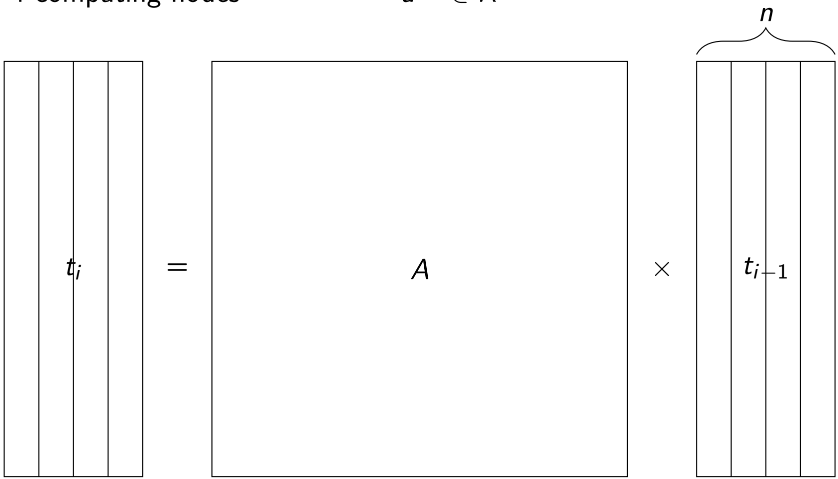
Parallelization of BW1 – cluster system

4 computing nodes

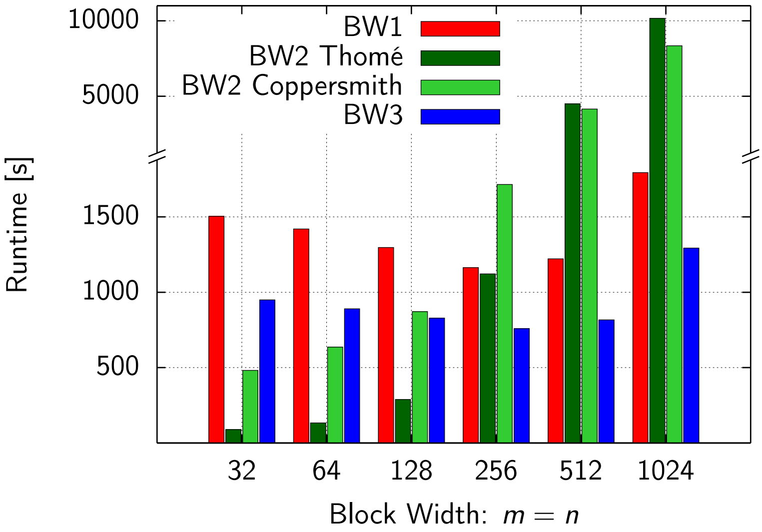


Parallelization of BW1 – cluster system

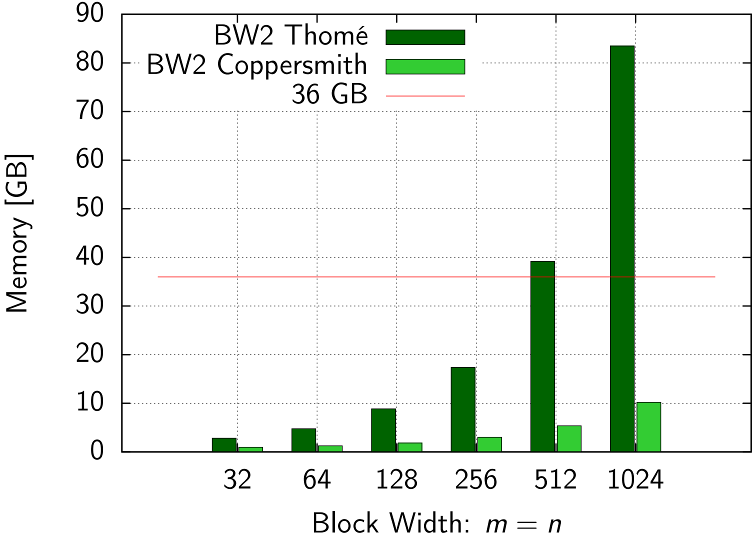
4 computing nodes $\qquad a^{(i)} \in K^{n \times m}$



Parallelization of BW1 – cluster system



Parallelization of BW1 – cluster system

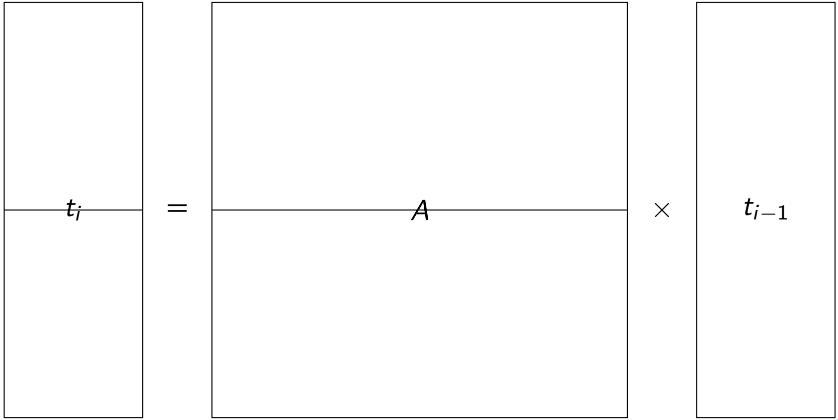


Parallelization of BW1 – cluster system

$$t_i = A \times t_{i-1}$$

Parallelization of BW1 – cluster system

2 computing nodes



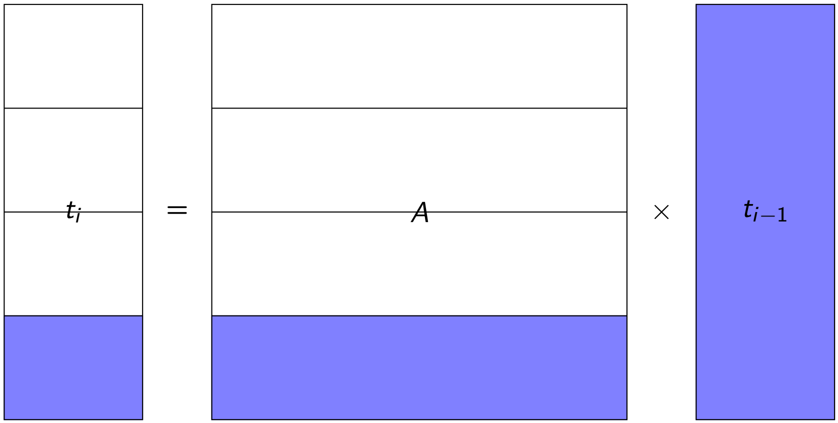
Parallelization of BW1 – cluster system

4 computing nodes

$$t_i = A \times t_{i-1}$$

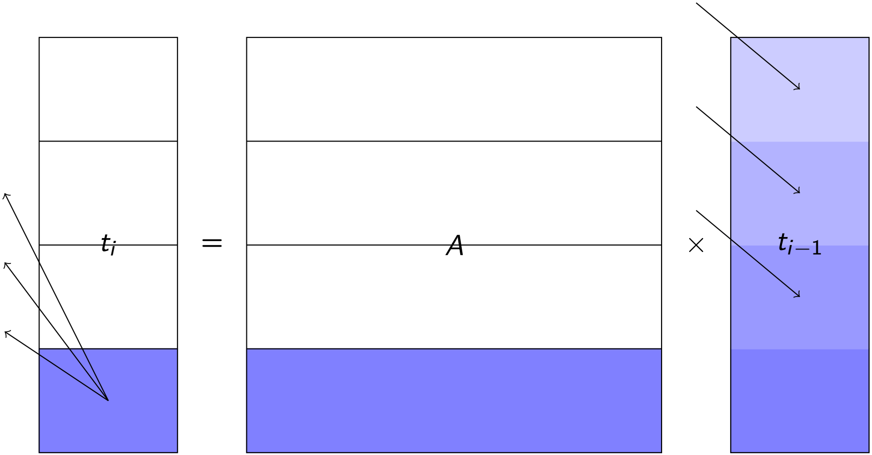
Parallelization of BW1 – cluster system

4 computing nodes



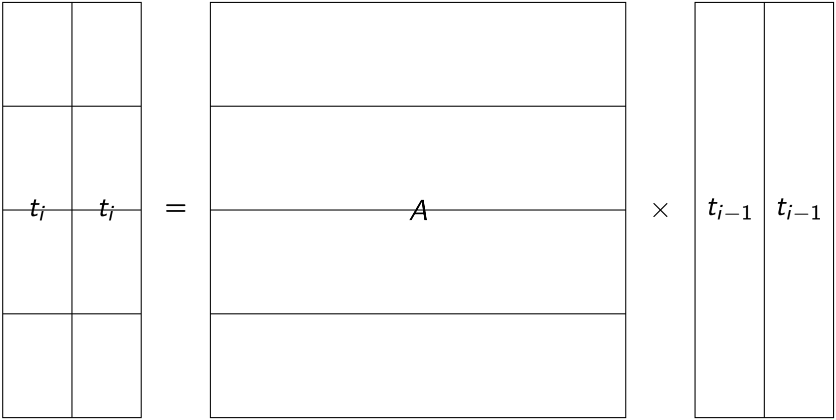
Parallelization of BW1 – cluster system

4 computing nodes



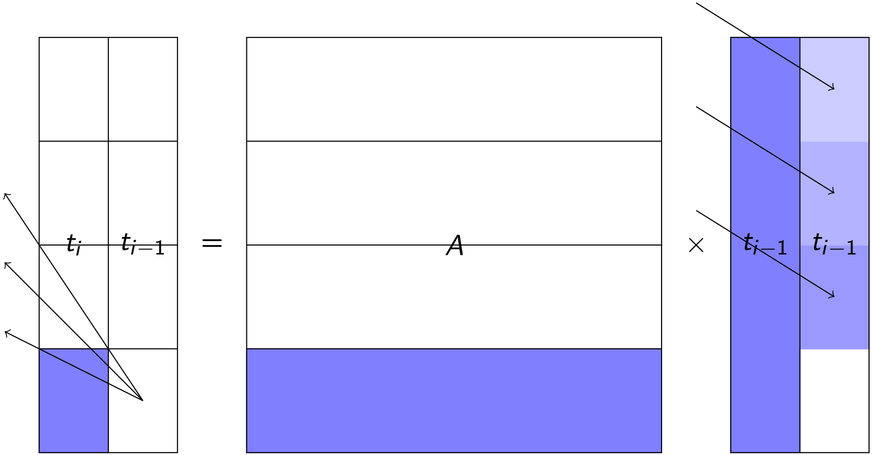
Parallelization of BW1 – cluster system

4 computing nodes



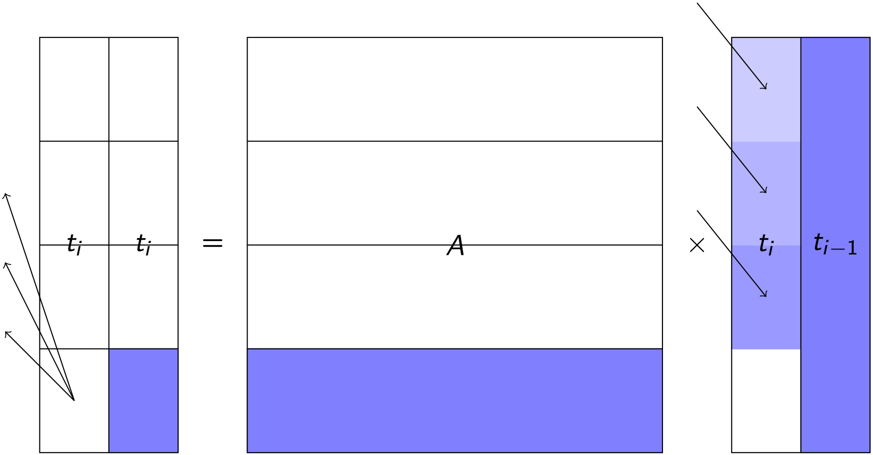
Parallelization of BW1 – cluster system

4 computing nodes



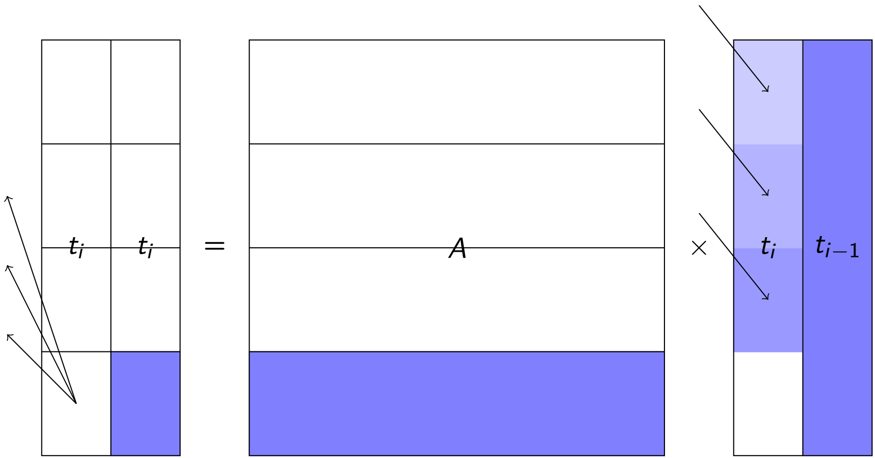
Parallelization of BW1 – cluster system

4 computing nodes



Parallelization of BW1 – cluster system

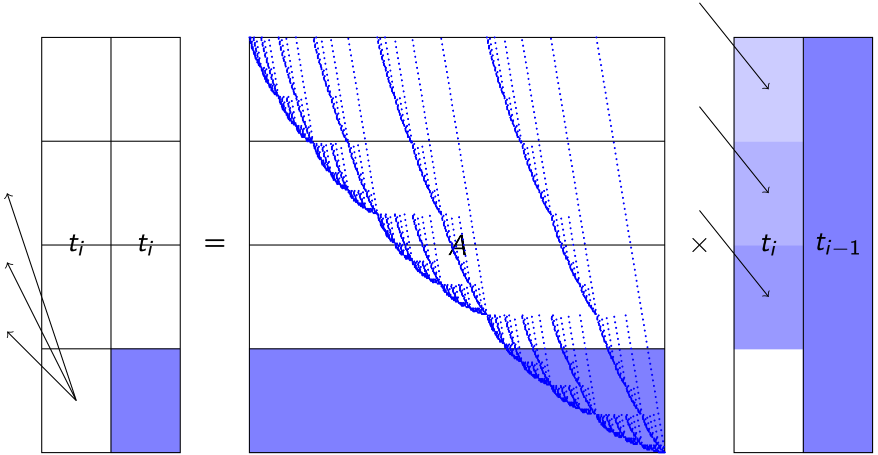
4 computing nodes



MPI: ISend, IRecv, ...

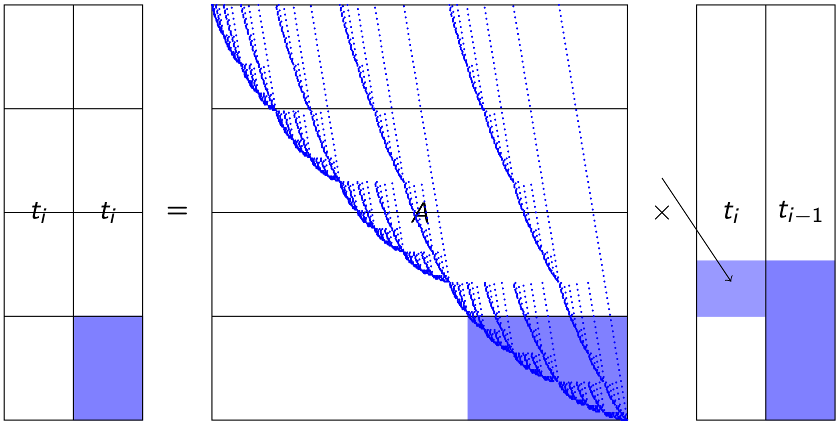
Parallelization of BW1 – cluster system

4 computing nodes



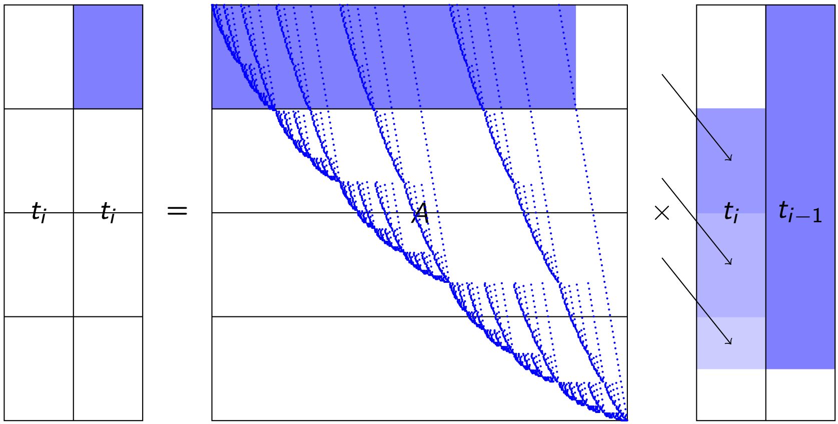
Parallelization of BW1 – cluster system

4 computing nodes



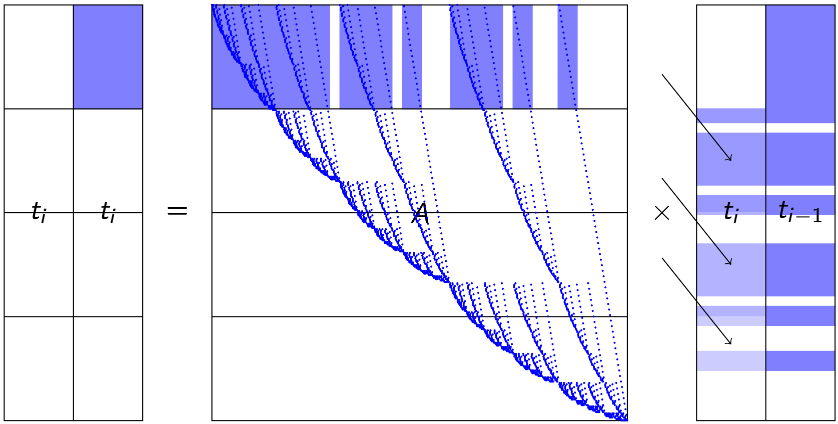
Parallelization of BW1 – cluster system

4 computing nodes



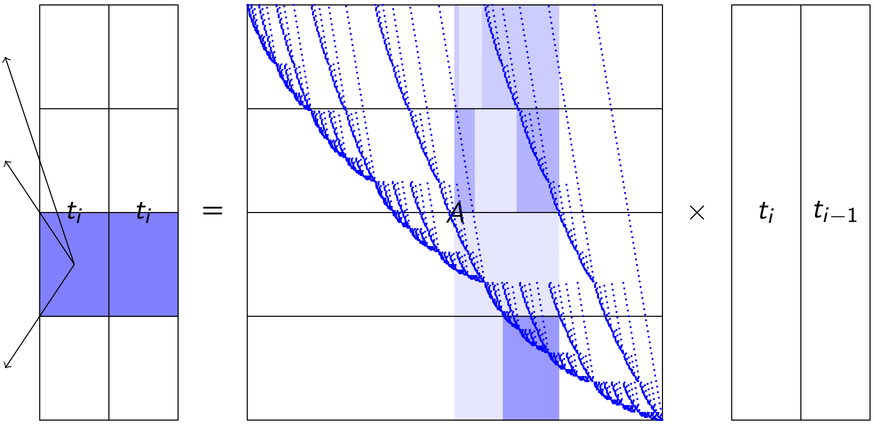
Parallelization of BW1 – cluster system

4 computing nodes



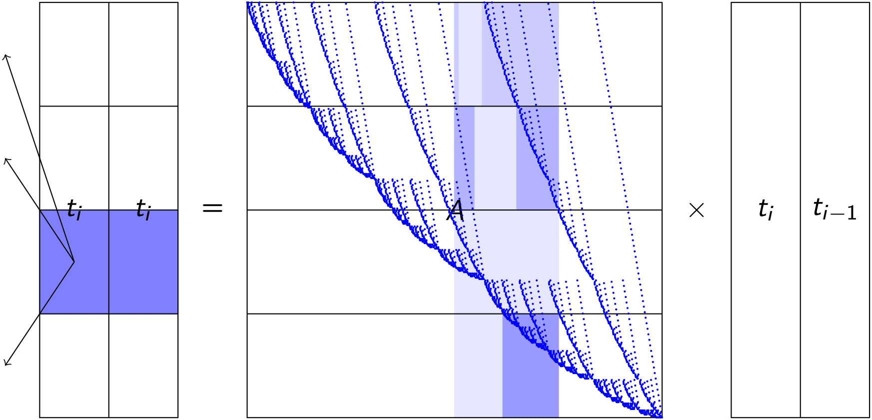
Parallelization of BW1 – cluster system

4 computing nodes



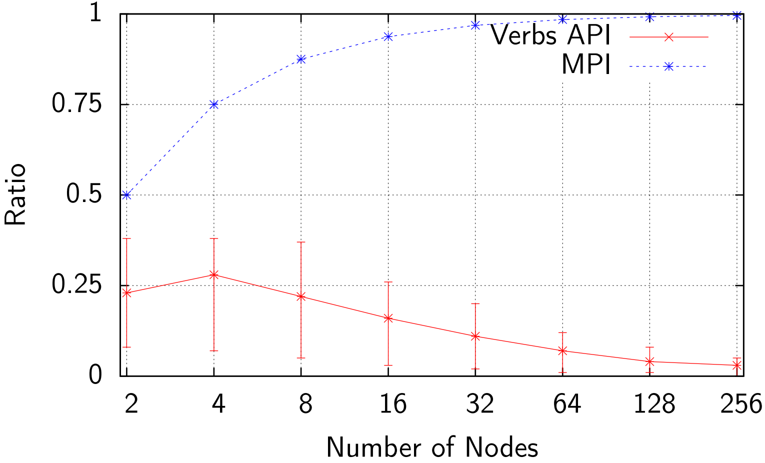
Parallelization of BW1 – cluster system

4 computing nodes

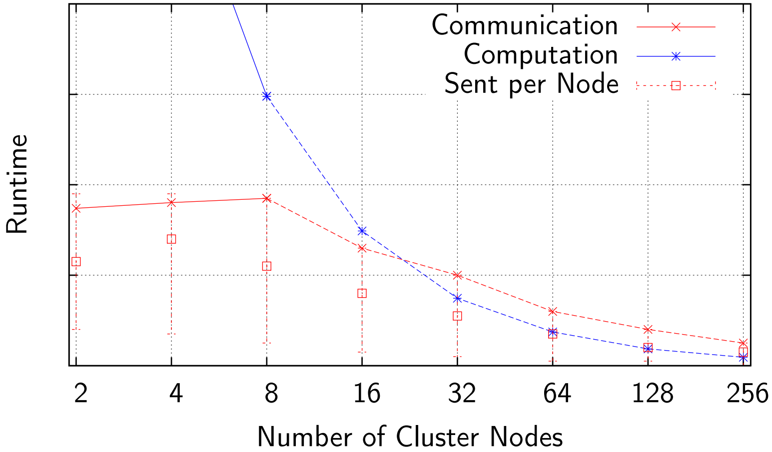


InfiniBand Verbs

Parallelization of BW1 – cluster system

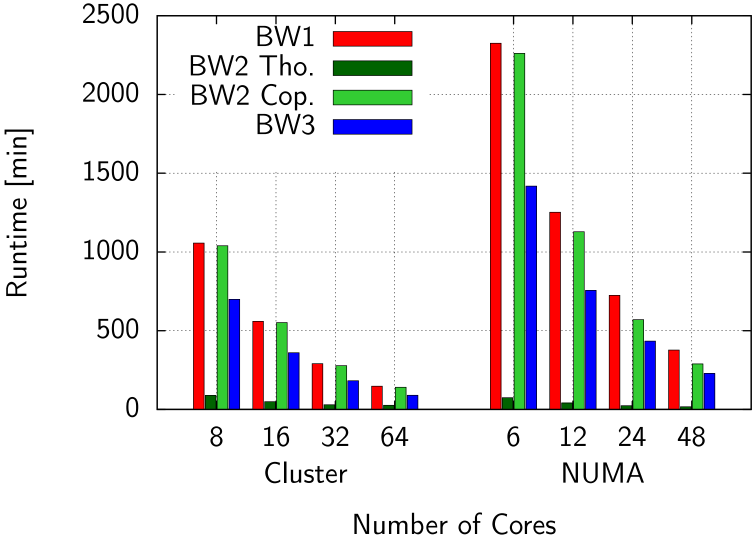


Parallelization of BW1 – cluster system

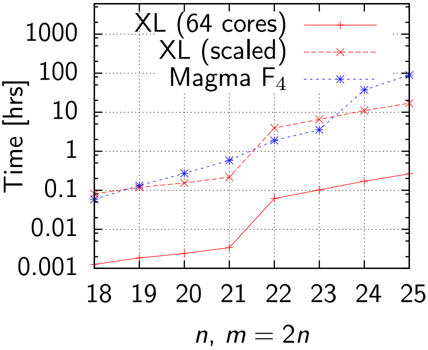


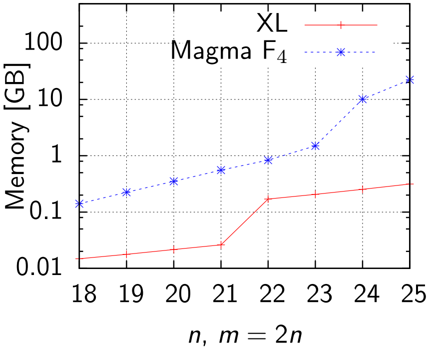
(InfiniBand MT26428, 2 ports of 4×QDR, 32 Gbit/s)

Runtime $n = 16$, $m = 18$, $\mathbb{F}_{16}$



Comparison to Magma F_4





Conclusions

XL with block Wiedemann as system solver is an alternative for Gröbner basis solvers, because

- in about 80% of the cases it operates on the same degree,
- it scales well on multicore systems and moderate cluster sizes,

and

- it has a relatively small memory demand.

Thank you!