

Solving Quadratic Equations with XL on Parallel Architectures

Cheng Chen-Mou¹, Chou Tung²,
Ni Ru-Ben², Yang Bo-Yin²

¹National Taiwan University

²Academia Sinica

Taipei, Taiwan

Leuven, Sept. 11, 2012

Solving Quadratic Equations with XL on Parallel Architectures

Chen-Mou Cheng¹, Tung Chou²,
Ruben Niederhagen², Bo-Yin Yang²

¹National Taiwan University

²Academia 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:
 $l_1 = l_2 = \dots = l_m = 0$, $l_i \in K[x_1, x_2, \dots, x_n]$
- choose: operational degree $D \in \mathbb{N}$
- extend: system \mathcal{A} to the system
 $\mathcal{R}^{(D)} = \{x^b l_i = 0 : |b| \leq D - 2, l_i \in \mathcal{A}\}$
- 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:
 $l_1 = l_2 = \dots = l_m = 0$, $l_i \in K[x_1, x_2, \dots, x_n]$
- choose: operational degree $D \in \mathbb{N}$ **How?**
- extend: system \mathcal{A} to the system
 $\mathcal{R}^{(D)} = \{x^b l_i = 0 : |b| \leq D - 2, l_i \in \mathcal{A}\}$
- 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:
 $l_1 = l_2 = \dots = l_m = 0$, $l_i \in K[x_1, x_2, \dots, x_n]$
- choose: operational degree $D \in \mathbb{N}$ How?
- extend: system \mathcal{A} to the system
 $\mathcal{R}^{(D)} = \{x^b l_i = 0 : |b| \leq D - 2, l_i \in \mathcal{A}\}$
- linearize: consider x^d , $d \leq D$ a new variable
to obtain a linear system \mathcal{M}
- solve: linear system \mathcal{M}

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 $x_1^{b_1} x_2^{b_2} \dots x_n^{b_n}$ 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:
 $l_1 = l_2 = \dots = l_m = 0$, $l_i \in K[x_1, x_2, \dots, x_n]$
- choose: operational degree $D \in \mathbb{N}$ **How?**
- extend: system \mathcal{A} to the system
 $\mathcal{R}^{(D)} = \{x^b l_i = 0 : |b| \leq D - 2, l_i \in \mathcal{A}\}$
- linearize: consider x^d , $d \leq D$ a new variable
to obtain a linear system \mathcal{M}
- solve: linear system \mathcal{M} **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 $x_1^{b_1} x_2^{b_2} \dots x_n^{b_n}$ 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:
 $\ell_1, \ell_2, \dots, \ell_m \in K[x_1, x_2, \dots, x_n]$

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

to obtain a linear system \mathcal{M}
solve: 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 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

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

Given a linearly recurrent sequence

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

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

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

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 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 $z \in K^{N \times n}$

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

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 $z \in K^{N \times n}$

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

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 $z \in K^{N \times n}$

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

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 :

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

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
```

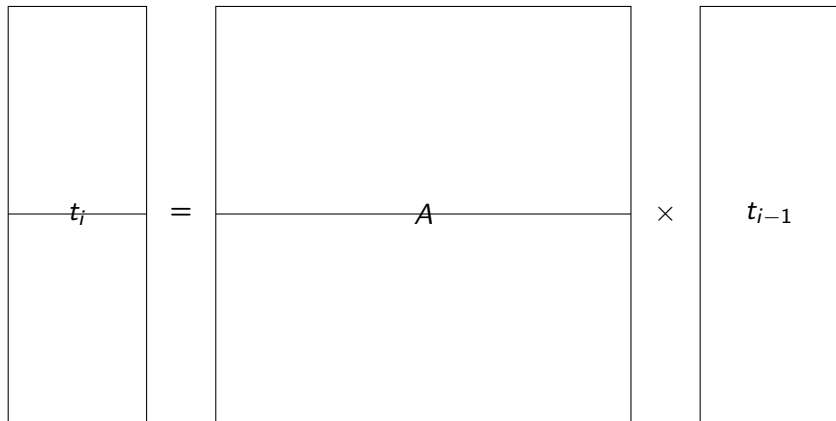
Parallelization of BW1 – multicore processor

The diagram illustrates the equation $t_i = A \times t_{i-1}$. Each term is enclosed in a rectangular box: t_i in a tall, narrow box on the left; A in a large square box in the center; and t_{i-1} in a tall, narrow box on the right. The terms are separated by an equals sign (=) and a multiplication sign (×).

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

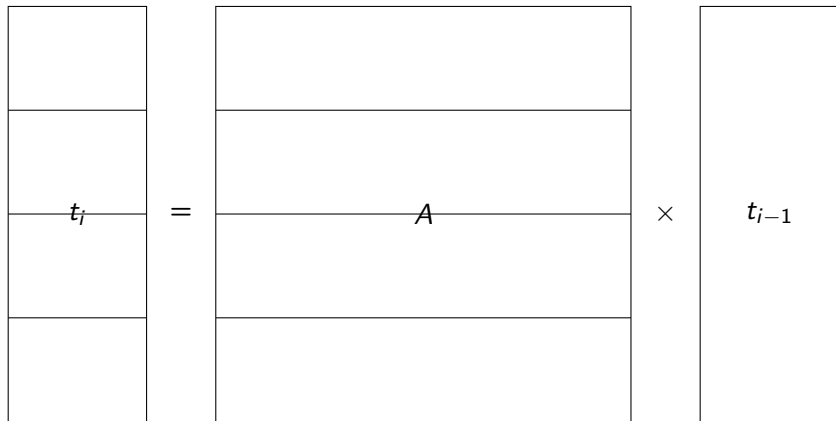
Parallelization of BW1 – multicore processor

2 cores



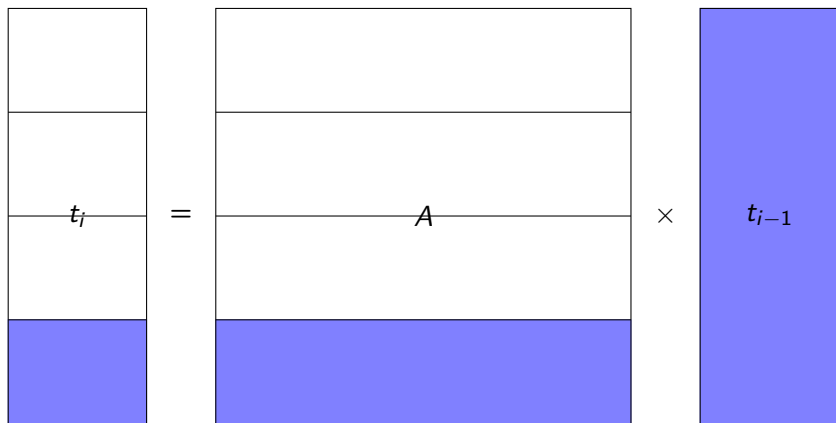
Parallelization of BW1 – multicore processor

4 cores



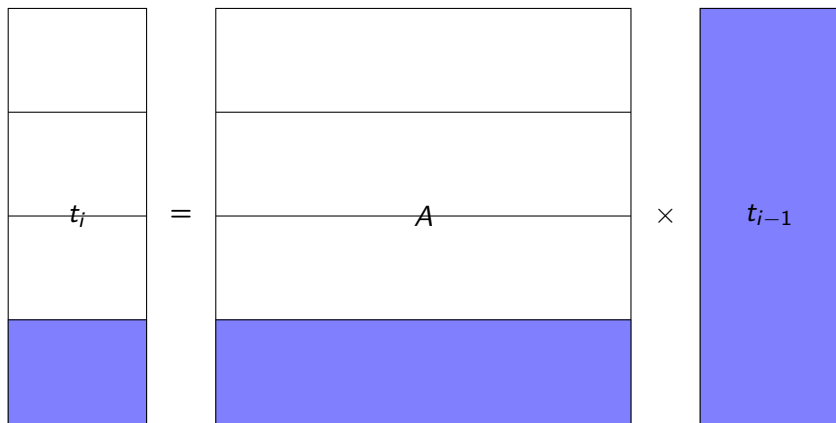
Parallelization of BW1 – multicore processor

4 cores



Parallelization of BW1 – multicore processor

4 cores



OpenMP

Parallelization of BW1 – cluster system

The diagram illustrates the equation $t_i = A \times t_{i-1}$. It features three rectangular boxes: a tall vertical box on the left containing the variable t_i , a large square box in the center containing the matrix A , and another tall vertical box on the right containing the variable t_{i-1} . The boxes are arranged horizontally, with an equals sign between the first and second boxes, and a multiplication sign between the second and third boxes.

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

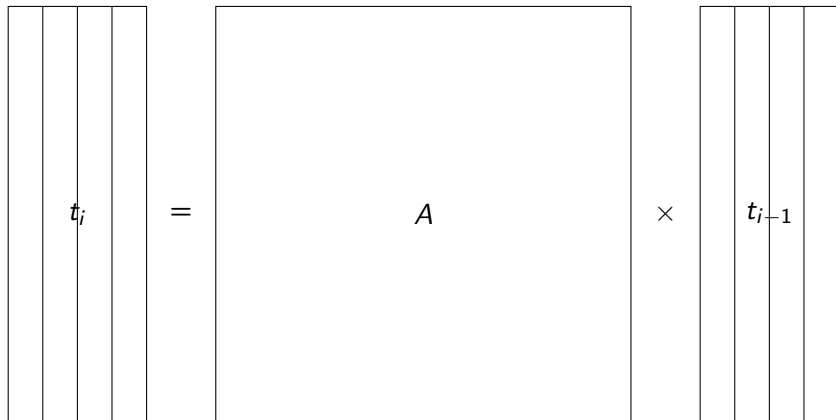
Parallelization of BW1 – cluster system

2 computing nodes

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

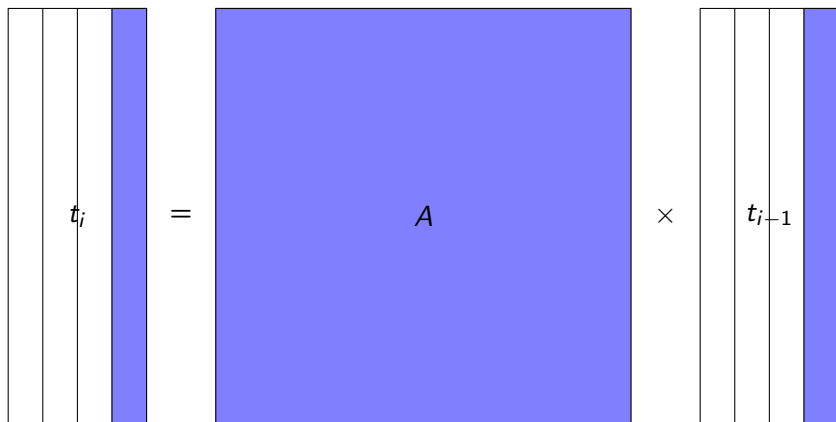
Parallelization of BW1 – cluster system

4 computing nodes



Parallelization of BW1 – cluster system

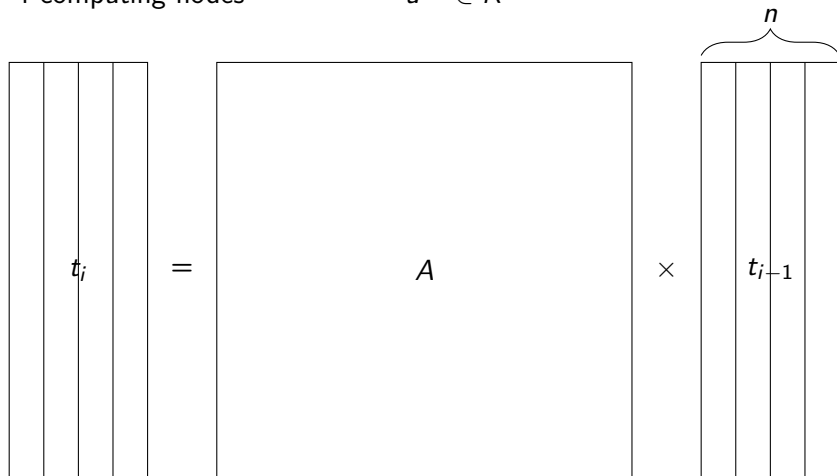
4 computing nodes



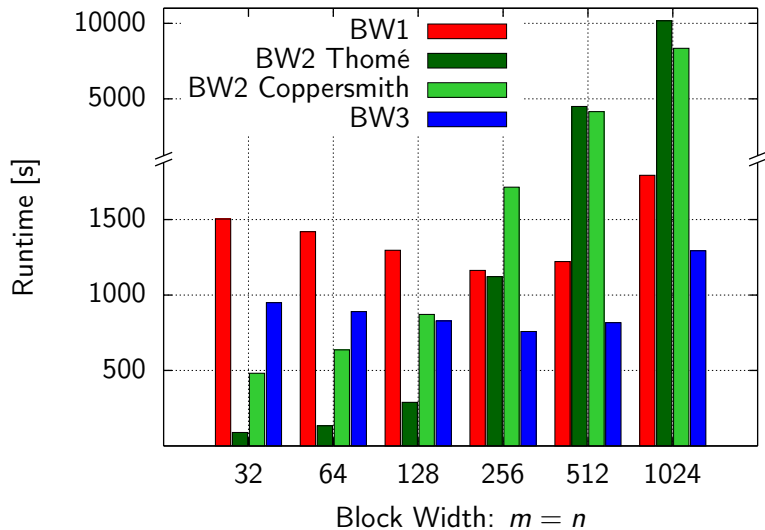
Parallelization of BW1 – cluster system

4 computing nodes

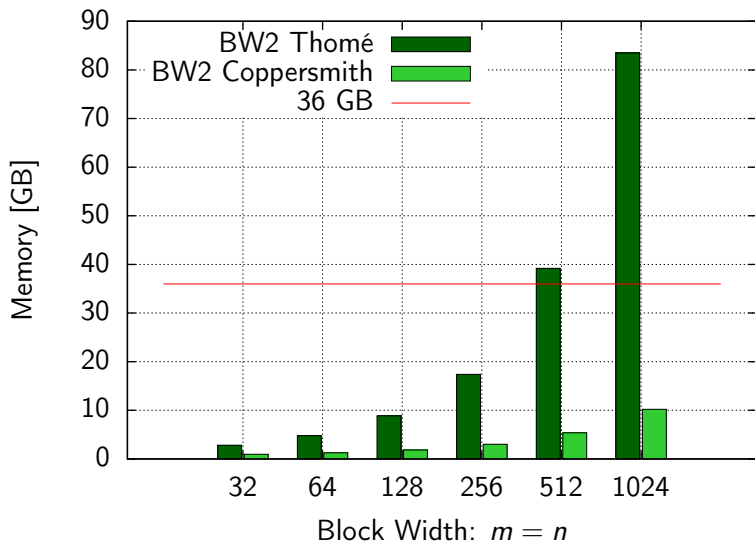
$$a^{(i)} \in K^{n \times m}$$



Parallelization of BW1 – cluster system



Parallelization of BW1 – cluster system



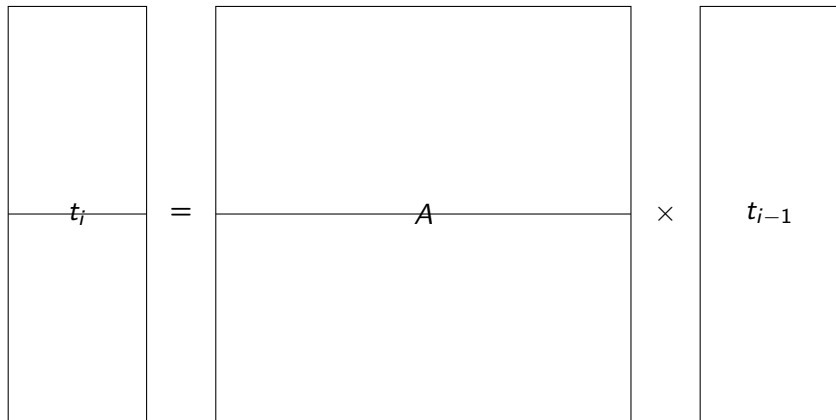
Parallelization of BW1 – cluster system

The diagram illustrates the equation $t_i = A \times t_{i-1}$. It features three rectangular boxes: a tall vertical box on the left containing the variable t_i , a large square box in the center containing the matrix A , and another tall vertical box on the right containing the variable t_{i-1} . The boxes are arranged horizontally, with an equals sign between the first and second boxes, and a multiplication sign between the second and third boxes.

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

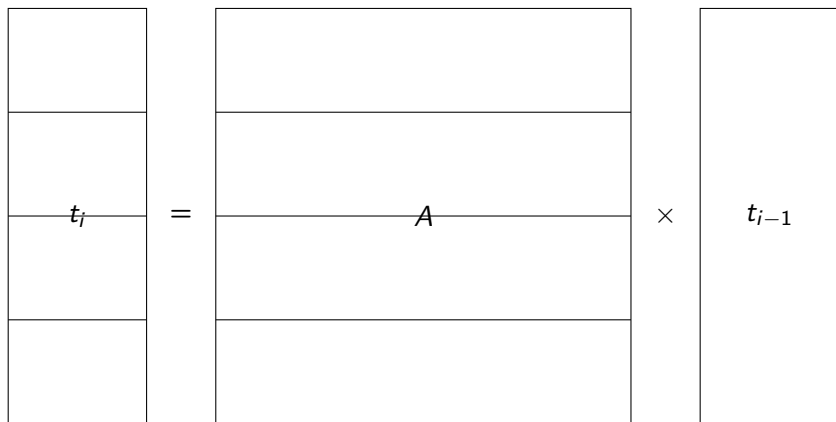
Parallelization of BW1 – cluster system

2 computing nodes



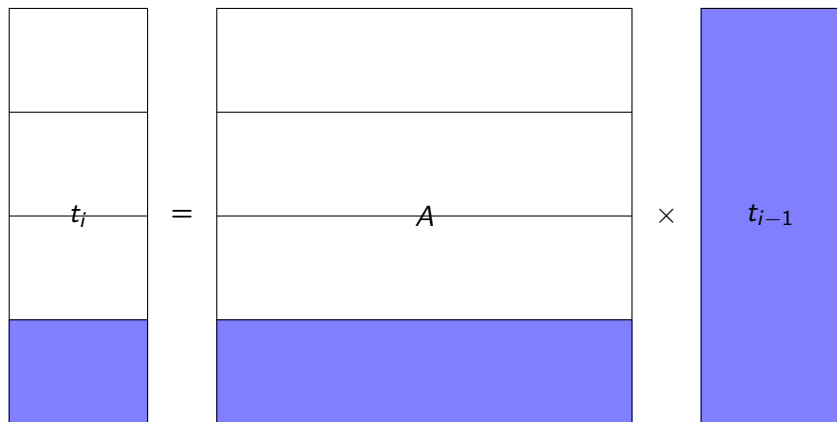
Parallelization of BW1 – cluster system

4 computing nodes



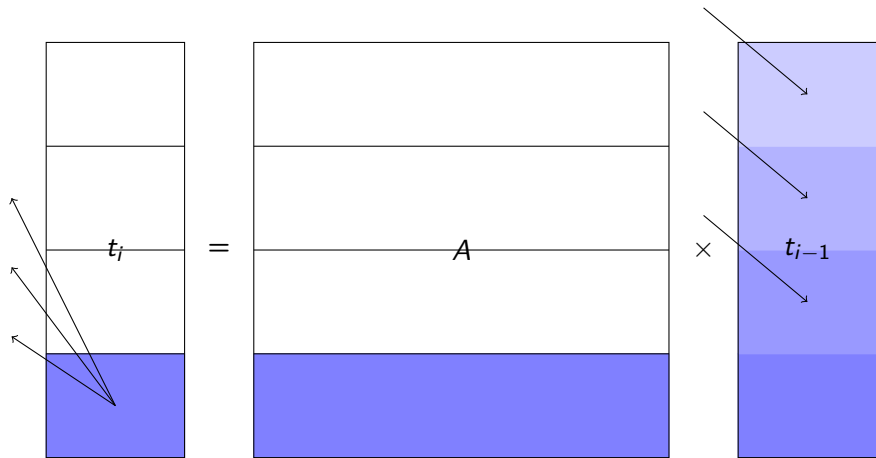
Parallelization of BW1 – cluster system

4 computing nodes



Parallelization of BW1 – cluster system

4 computing nodes



Parallelization of BW1 – cluster system

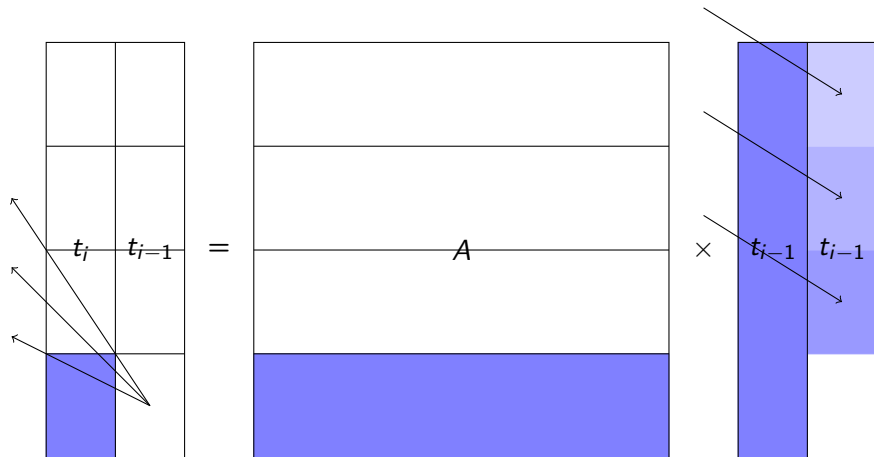
4 computing nodes

The diagram illustrates the parallelization of a matrix-vector multiplication on 4 computing nodes. It shows the following components:

- Left Matrix:** A 4x2 grid representing the result vector. The middle two rows are labeled t_j and t_j , indicating that each of the 4 nodes is responsible for one of these two values.
- Equality Sign:** A central equals sign ($=$) indicating the operation.
- Middle Matrix:** A 4x1 grid representing the matrix A . The middle row is labeled A , indicating that the matrix is distributed across the 4 nodes.
- Multiplication Sign:** A central multiplication sign (\times) indicating the operation.
- Right Matrix:** A 4x2 grid representing the input vector. The middle two rows are labeled t_{i-1} and t_{i-1} , indicating that each of the 4 nodes is responsible for one of these two values.

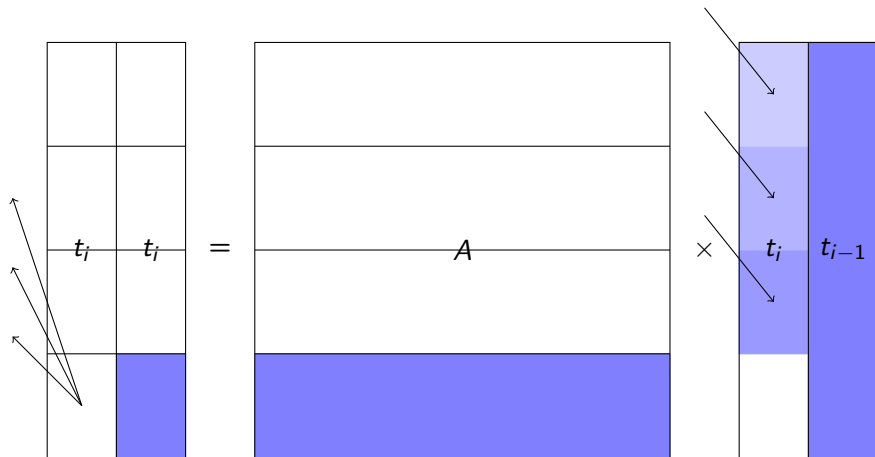
Parallelization of BW1 – cluster system

4 computing nodes



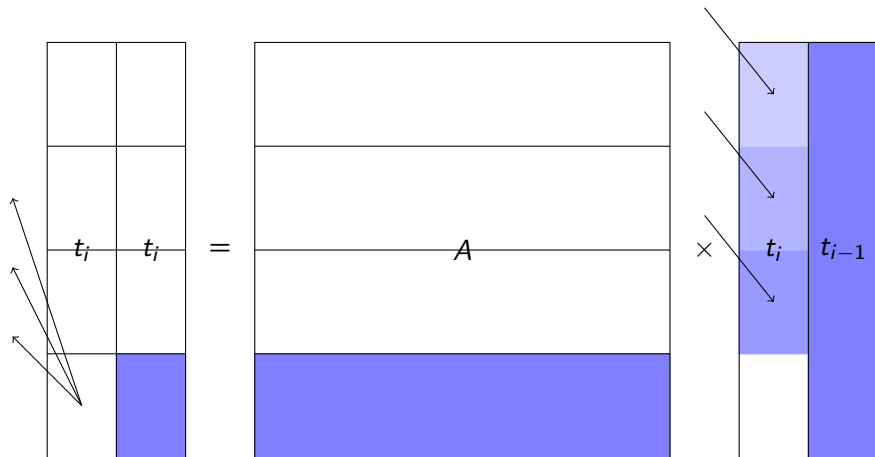
Parallelization of BW1 – cluster system

4 computing nodes



Parallelization of BW1 – cluster system

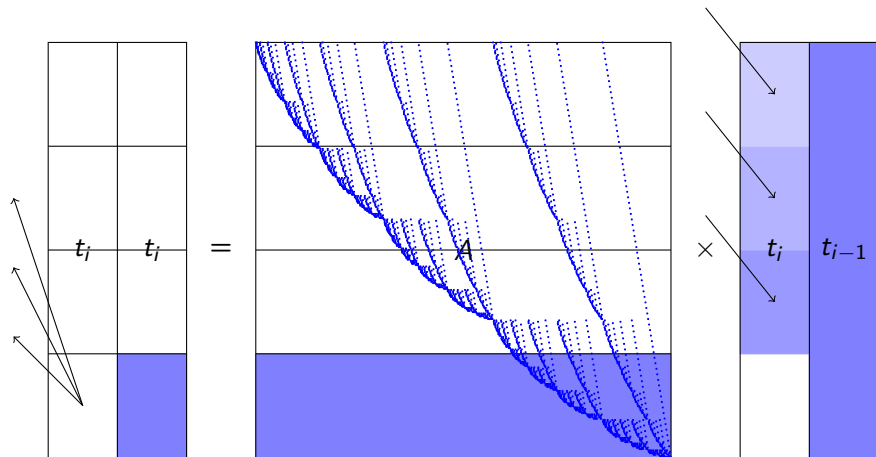
4 computing nodes



MPI: ISend, IRecv, ...

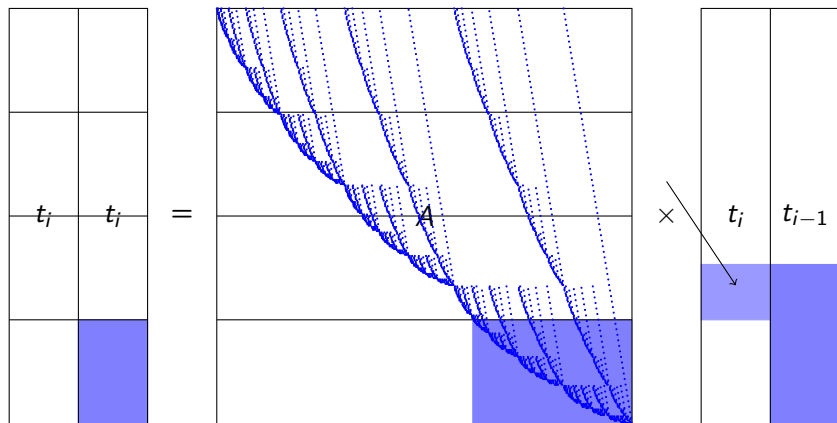
Parallelization of BW1 – cluster system

4 computing nodes



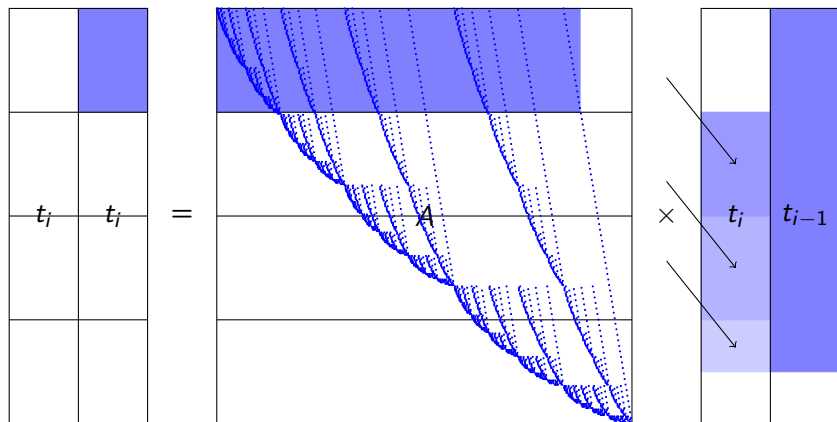
Parallelization of BW1 – cluster system

4 computing nodes



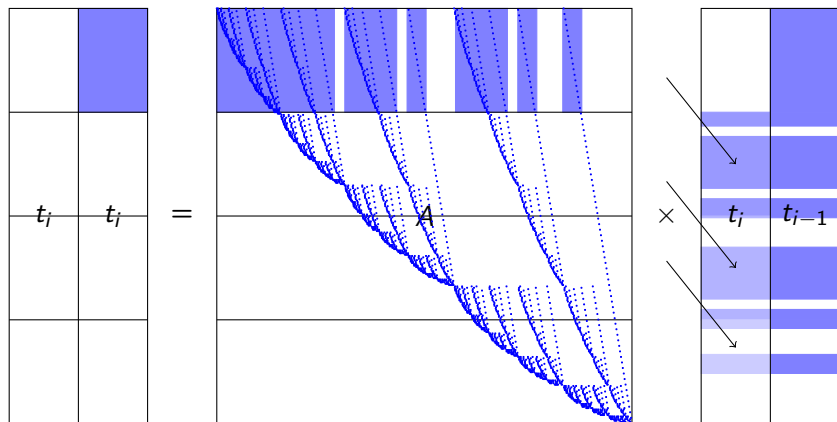
Parallelization of BW1 – cluster system

4 computing nodes



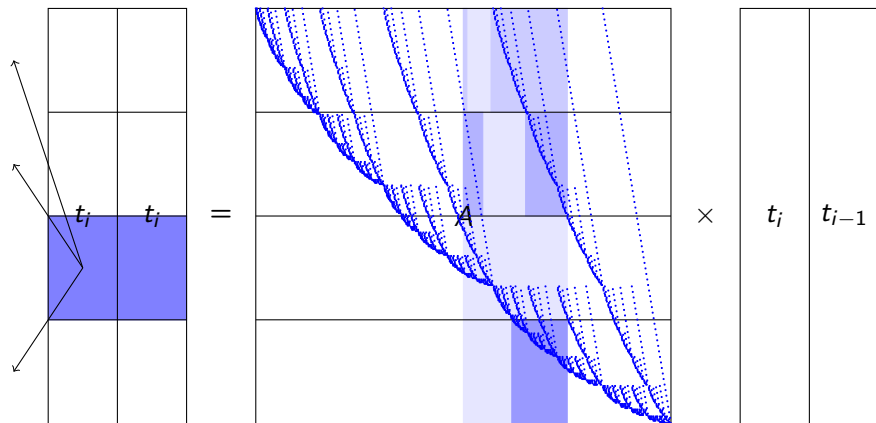
Parallelization of BW1 – cluster system

4 computing nodes



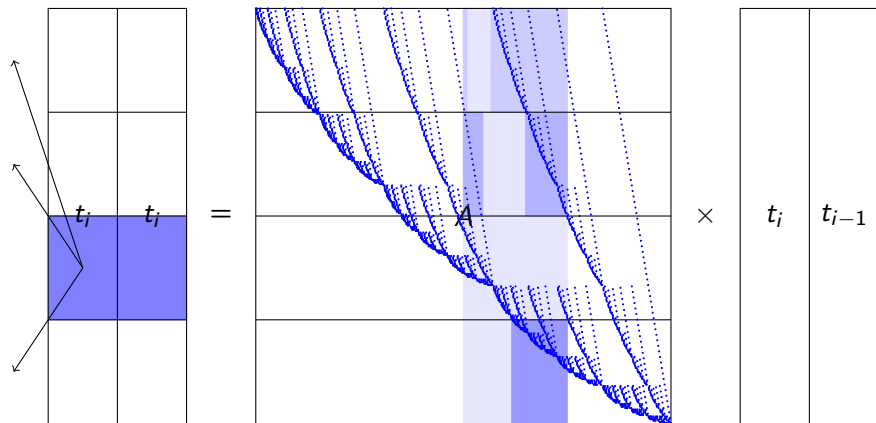
Parallelization of BW1 – cluster system

4 computing nodes



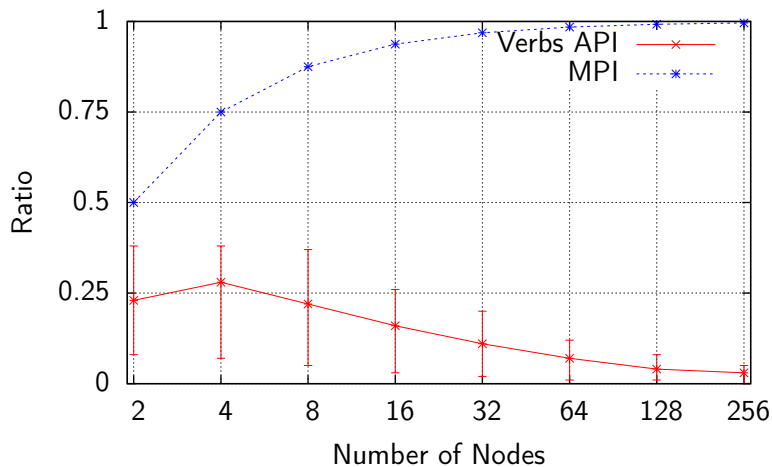
Parallelization of BW1 – cluster system

4 computing nodes

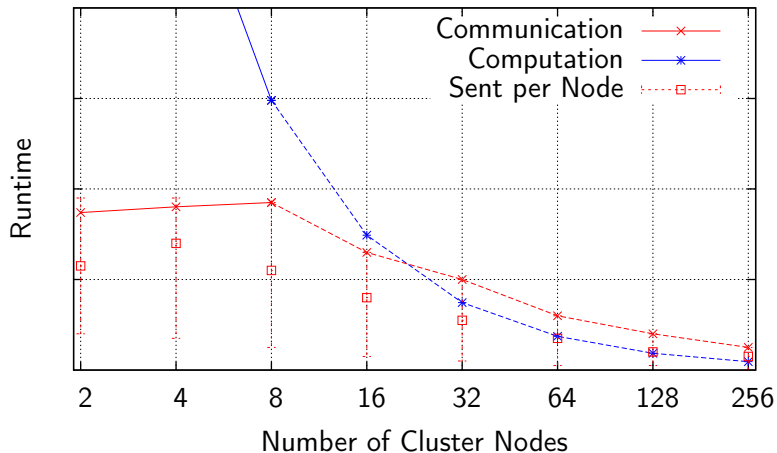


InfiniBand Verbs

Parallelization of BW1 – cluster system

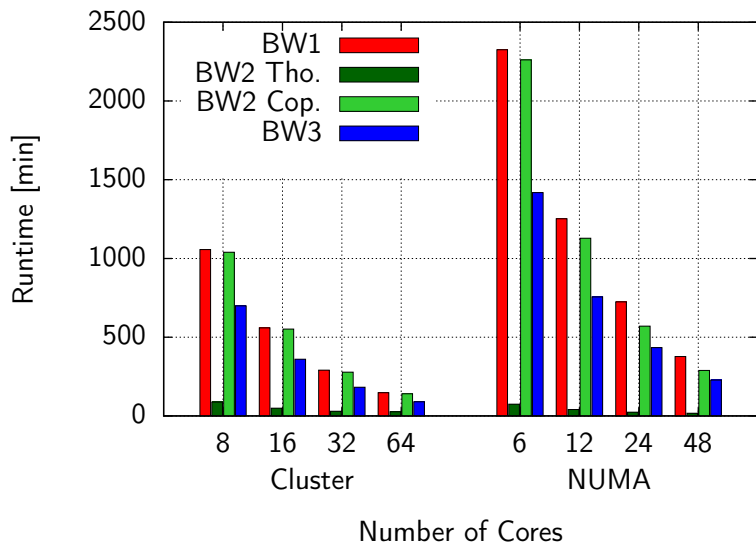


Parallelization of BW1 – cluster system

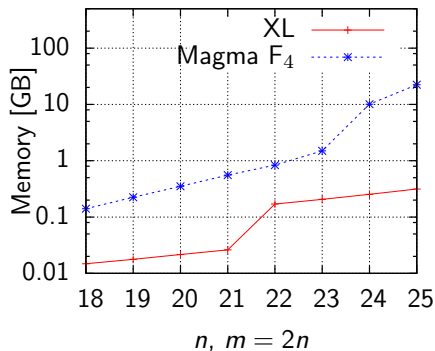
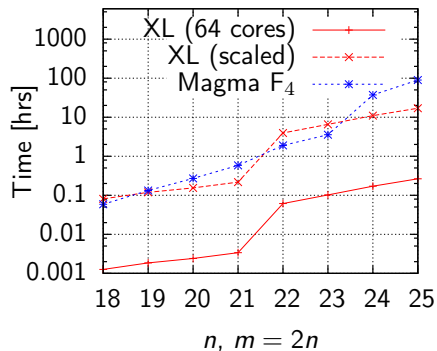


(InfiniBand MT26428, 2 ports of 4×QDR, 32 Gbit/s)

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!