Efficient Modular Matrix Multiplication on GPU for Polynomial System solving

Jérémy Berthomieu, Stef Graillat, **Dimitri Lesnoff**, Theo Mary

LIP6 Sorbonne Université CNRS

Journées Nationales de Calcul Formel 6 – 10 mars 2023

Exact Polynomial System Solving

We want to solve **exactly** polynomial systems
For polynomial ideals of dimension 0

Polynomial System

$x_1, \ldots, x_n$ unknowns,
$f_1, \ldots, f_m$ polynomials

$$\begin{cases} f_1(x_1, \ldots, x_n) & = 0 \\ \vdots & = \vdots \\ f_m(x_1, \ldots, x_n) & = 0 \end{cases}$$

$\rightarrow$

Exact Polynomial System Solving

We want to solve **exactly** polynomial systems
For polynomial ideals of dimension 0

Polynomial System

$x_1, \ldots, x_n$ unknowns,
$f_1, \ldots, f_m$ polynomials

$$\begin{cases} f_1(x_1, \ldots, x_n) &= 0 \\ \vdots & = \vdots \\ f_m(x_1, \ldots, x_n) &= 0 \end{cases}$$

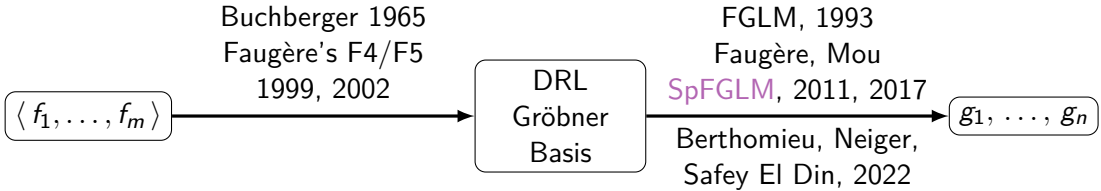
$\rightarrow$

LEX Gröbner Basis

Shape Position

$$\begin{cases} g_n(x_n) &= 0 \\ x_{n-1} &= g_{n-1}(x_n) \\ \vdots & = \vdots \\ x_1 &= g_1(x_n) \end{cases}$$

Gröbner basis computation



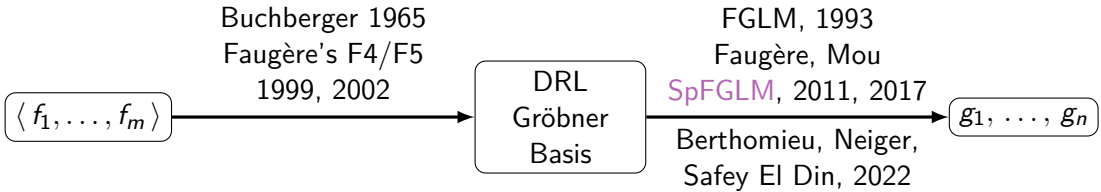
SpFGLM with Block-Wiedemann

- Matrix sequence computation:

$$\boldsymbol{r} \xrightarrow{M\cdot} M\boldsymbol{r} \xrightarrow{M\cdot} M^2\boldsymbol{r} \xrightarrow{M\cdot} \ldots \xrightarrow{M\cdot} M^{2D/s-1}\boldsymbol{r}, \quad M \in \mathbb{K}^{D\times D}, \boldsymbol{r} \in \mathbb{K}^{D\times s}$$

- Matrix linear recurrence guessing (PML) [Hyun,Neiger,Schost 2019]

Gröbner basis computation



SpFGLM with Block-Wiedemann

- Matrix sequence computation:

$$\boldsymbol{r} \xrightarrow{M\cdot} M\boldsymbol{r} \xrightarrow{M\cdot} M^2\boldsymbol{r} \xrightarrow{M\cdot} \ldots \xrightarrow{M\cdot} M^{2D/s-1}\boldsymbol{r}, \quad M \in \mathbb{K}^{D\times D}, \boldsymbol{r} \in \mathbb{K}^{D\times s}$$

- Matrix linear recurrence guessing (PML) [Hyun,Neiger,Schost 2019]

Challenge

Matrix Multiplication: $M' \cdot \boldsymbol{r}, \quad M' \in \mathbb{F}_p^{t \times D}, \mathbb{F}_p^{D \times s}$

Field: $\mathbb{K} = \mathbb{F}_p$, p large prime

Matrix sizes for benchmarks: Target: $D \simeq 100\,000$, p more than 26 bits

Objective: Integrate GPU matrix multiplication in MSOLVE

Plan of this talk

1. Reduction with floating-point: FMA (Fused Multiply-Add) instruction
2. Dot-Product with DMR (Delayed Modular Reduction)
3. Matrix product over finite field (Single Word, Multi Word)

Challenge

Matrix Multiplication: $M' \cdot \boldsymbol{r}, \quad M' \in \mathbb{F}_p^{t \times D}, \mathbb{F}_p^{D \times s}$

Field: $\mathbb{K} = \mathbb{F}_p$, p large prime

Matrix sizes for benchmarks: Target: $D \simeq 100\,000$, p more than 26 bits

Objective: Integrate GPU matrix multiplication in MSOLVE

Plan of this talk

1. Reduction with floating-point: FMA (Fused Multiply-Add) instruction
2. Dot-Product with DMR (Delayed Modular Reduction)
3. Matrix product over finite field (Single Word, Multi Word)

Finite Field Matrix Multiplication

Existing CPU libraries

- NTL [Shoup 2002] (zz_p)
- FLINT[Hart, Johansson, Pancratz, 2007]
  (`nmod_mat_mul` for single word prime fields, `fmpz_mod_mat_mul` for multi-word)
- FFLAS-FFPACK [Dumas, Giorgi, Pernet, 2008]

Existing GPU libraries

- MAGMA Computational Algebra Group, University of Sydney

All libraries use floating-point representation. Prime limit:

integer: 2^{64} limit $\rightarrow$ for multiplication $p < 2^{32}$

float: 2^{53} limit $\rightarrow$ for multiplication $p < 2^{26}$

Scarce native support for 64 bits integers on GPU

Finite Field Matrix Multiplication

Existing CPU libraries

- NTL [Shoup 2002] (zz_p) Multithreading
- FLINT[Hart, Johansson, Pancratz, 2007] Multithreading OR Strassen
  (`nmod_mat_mul` for single word prime fields, `fmpz_mod_mat_mul` for multi-word)
- FFLAS-FFPACK [Dumas, Giorgi, Pernet, 2008] Multithreading AND Strassen

Existing GPU libraries

- MAGMA Computational Algebra Group, University of Sydney

All libraries use floating-point representation. Prime limit:

integer: 2^{64} limit $\rightarrow$ for multiplication $p < 2^{32}$

float: 2^{53} limit $\rightarrow$ for multiplication $p < 2^{26}$

Scarce native support for 64 bits integers on GPU

Finite Field Matrix Multiplication

Existing CPU libraries

- NTL [Shoup 2002] (zz_p) Multithreading
- FLINT[Hart, Johansson, Pancratz, 2007] Multithreading OR Strassen
  (`nmod_mat_mul` for single word prime fields, `fmpz_mod_mat_mul` for multi-word)
- FFLAS-FFPACK [Dumas, Giorgi, Pernet, 2008] Multithreading AND Strassen

Existing GPU libraries

- MAGMA Computational Algebra Group, University of Sydney

All libraries use floating-point representation. Prime limit:

integer: 2^{64} limit $\rightarrow$ for multiplication $p < 2^{32}$

float: 2^{53} limit $\rightarrow$ for multiplication $p < 2^{26}$

Scarce native support for 64 bits integers on GPU

Finite Field Matrix Multiplication

Existing CPU libraries

- NTL [Shoup 2002] (zz_p) Multithreading
- FLINT[Hart, Johansson, Pancratz, 2007] Multithreading OR Strassen
  (`nmod_mat_mul` for single word prime fields, `fmpz_mod_mat_mul` for multi-word)
- FFLAS-FFPACK [Dumas, Giorgi, Pernet, 2008] Multithreading AND Strassen

Existing GPU libraries

- MAGMA Computational Algebra Group, University of Sydney

All libraries use floating-point representation. Prime limit:

integer: 2^{64} limit $\rightarrow$ for multiplication $p < 2^{32}$

float: 2^{53} limit $\rightarrow$ for multiplication $p < 2^{26}$

Scarce native support for 64 bits integers on GPU

Modular Reduction with FMA

- No modulo operator for floating-point type.
  "real" division, NTL [Shoup 2002]
  → Residue is correct up to $\pm p$ for $p < 2^{26}$

$$x \bmod p \equiv \mathrm{fl}\left(x - \left\lfloor \frac{x}{p} \right\rfloor \times p\right) \in [-p, 2p-1]$$

- Better reduction with Fused Multiply-Add instruction
  [Jean, Graillat 2010] [van der Hoeven, Lecerf, Quintin 2014] (Mathemagix)
  → Approximated addition / Exact multiplication
  → Reduction is exact for $p < 2^{50}$ and $x \leq 2^{50} * p$
  → Simplest use of Error-Free Transformation (EFT)

Modular Reduction with FMA

- No modulo operator for floating-point type.
  "real" division, NTL [Shoup 2002]
  → Residue is correct up to $\pm p$ for $p < 2^{26}$

$$x \bmod p \equiv \underbrace{\mathrm{fl}\left(x - \left\lfloor \frac{x}{p} \right\rfloor \times p\right)}_{\texttt{fma}(-c,\, p,\, x)} \in [-p, 2p-1]$$

- Better reduction with Fused Multiply-Add instruction
  [Jean, Graillat 2010] [van der Hoeven, Lecerf, Quintin 2014] (Mathemagix)
  → Approximated addition / Exact multiplication
  → Reduction is exact for $p < 2^{50}$ and $x \leq 2^{50} * p$
  → Simplest use of Error-Free Transformation (EFT)

Delayed Modular Reduction

- Compute the **dot product** by reparenthesing into blocks of λ summands which does not require modular reduction $\rightarrow$ use `ddot` BLAS routine
- **Reduce** the sum **just before** overflow
- Optimal lambda [Dumas, Gautier, Pernet 2002]:

$$\lambda_{\text{opt}}(p-1)^2 < 2^{53} < (\lambda_{\text{opt}}+1)(p-1)^2 \quad ; \quad \lambda_{\text{opt}} = \left\lfloor 2^{53}/(p-1)^2 \right\rfloor$$

$$\langle u, v \rangle = (\overbrace{u_1}^{\leq (p-1)} v_1 + \cdots + u_\lambda v_\lambda) \bmod p + \cdots + (u_{D-\lambda+1} v_{D-\lambda+1} + \cdots + u_D v_D) \bmod p$$

Figure: Delayed Modular Reduction in a Dot Product

Delayed Modular Reduction

- Compute the **dot product** by reparenthesing into blocks of λ summands which does not require modular reduction $\rightarrow$ use `ddot` BLAS routine
- **Reduce** the sum **just before** overflow
- Optimal lambda [Dumas, Gautier, Pernet 2002]:

$$\lambda_{\text{opt}}(p-1)^2 < 2^{53} < (\lambda_{\text{opt}}+1)(p-1)^2 \quad ; \quad \lambda_{\text{opt}} = \left\lfloor 2^{53}/(p-1)^2 \right\rfloor$$

$$\langle u, v \rangle = \Big(\underbrace{\overbrace{u_1}^{\leq (p-1)} v_1}_{\leq (p-1)^2} + \cdots + \underbrace{u_\lambda v_\lambda}_{\leq (p-1)^2} \Big)_{\leq \lambda\cdot(p-1)^2 \,<\, 2^{53}} \bmod p + \cdots + \Big(\underbrace{u_{D-\lambda+1} v_{D-\lambda+1}}_{\leq (p-1)^2} + \cdots + \underbrace{u_D v_D}_{\leq (p-1)^2} \Big)_{\leq \lambda\cdot(p-1)^2 \,<\, 2^{53}} \bmod p$$

Figure: Delayed Modular Reduction in a Dot Product

Delayed Modular Reduction

- Compute the **dot product** by reparenthesing into blocks of λ summands which does not require modular reduction $\rightarrow$ use `ddot` BLAS routine
- **Reduce** the sum **just before** overflow
- Optimal lambda [Dumas, Gautier, Pernet 2002]:

$$\lambda_{\text{opt}}(p-1)^2 < 2^{53} < (\lambda_{\text{opt}}+1)(p-1)^2 \quad ; \quad \lambda_{\text{opt}} = \left\lfloor 2^{53}/(p-1)^2 \right\rfloor$$

$$\langle u, v\rangle = \Big(\underbrace{\overbrace{u_1}^{\leq (p-1)} \; v_1}_{\leq (p-1)^2} + \cdots + \underbrace{u_\lambda v_\lambda}_{\leq (p-1)^2} \Big)_{\leq \lambda\cdot(p-1)^2 \,<\, 2^{53}} \bmod p + \cdots + \Big(\underbrace{u_{D-\lambda+1} v_{D-\lambda+1}}_{\leq (p-1)^2} + \cdots + \underbrace{u_D v_D}_{\leq (p-1)^2} \Big)_{\leq \lambda\cdot(p-1)^2 \,<\, 2^{53}} \bmod p$$

$$\leq \left\lfloor \tfrac{N}{\lambda} \right\rfloor p \text{ Additional reductions when } \geq 2^{53}$$

Figure: Delayed Modular Reduction in a Dot Product

Matrix Multiplication in prime fields

ffMatMul in CUBLAS

Input: $A \in \mathbb{F}^{t \times D}$ $B \in \mathbb{F}^{D \times s}$ matrices (double precision)

Output: $C = AB$

Set $\lambda = \left\lfloor 2^{53}/(p-1)^2 \right\rfloor$. For j from 1 to $\lfloor N/\lambda \rfloor$ do:

1. Multiply $A_j \in \mathbb{F}^{t \times \lambda}$ and $B_j \in \mathbb{F}^{\lambda \times s}$ and store product in temporary T.
2. Reduce T coefficient by coefficient using FMA
3. Store partial result T in resulting matrix C (buffer).

Reduce buffer C (coeff. by coeff.) and return it.

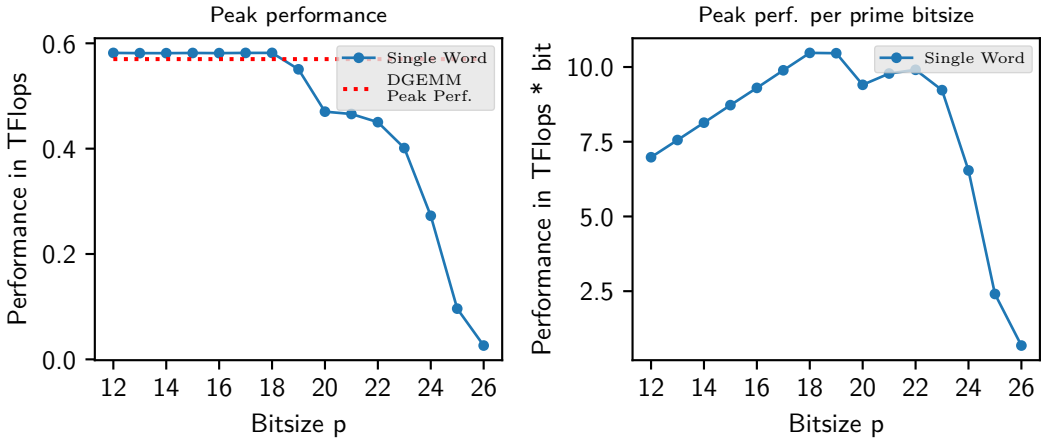


Figure: Performance of Single Word matrix product on A40 GPU (t:15000, D:45000, s:32) F_p: prime field

Multiword computation

Two Word Matrix Multiplication

$$A = 2^{t/2} \cdot A_h + A_l \quad A_h,\ A_l \in \mathbb{F}^{t \times D}$$
$$B = 2^{t/2} \cdot B_h + B_l \quad B_h,\ B_l \in \mathbb{F}^{D \times s}$$

$$\text{(Naive } 4\times) \quad C = A \cdot B = 2^t \cdot A_h \cdot B_h + 2^{t/2} \cdot (A_h \cdot B_l + A_l \cdot B_h) + A_l \cdot B_l$$

Drawback: greater memory consumption, depends additionally from the inner dimension

- $O(ts)$ for SingleWord
- $O(tD + Ds + ts)$ for MultiWord

Multiword computation

Two Word Matrix Multiplication

$$A = 2^{t/2} \cdot A_h + A_l \quad A_h,\ A_l \in \mathbb{F}^{t \times D}$$
$$B = 2^{t/2} \cdot B_h + B_l \quad B_h,\ B_l \in \mathbb{F}^{D \times s}$$

$$\text{(Naive 4}\times\text{)} \quad C = A \cdot B = 2^t \cdot A_h \cdot B_h + 2^{t/2} \cdot (A_h \cdot B_l + A_l \cdot B_h) + A_l \cdot B_l$$

Advantages are threefold!

- Greater prime: We can target primes $p > 2^{26}$!
- Smaller precision can be used (with more than 2 words)
- Greater lambda limit per prime: $\lambda(p) \to \lambda(p/2)$

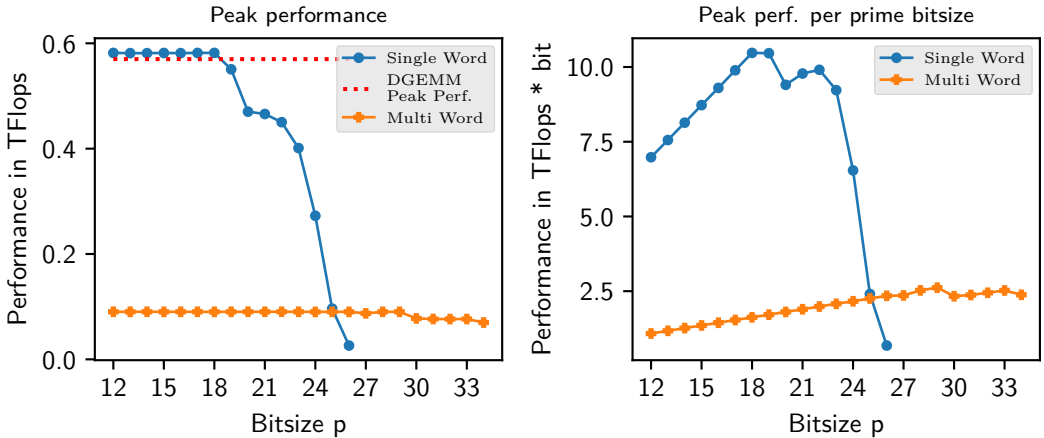


Figure: Comparison between Single and MultiWord matrix product on A40 GPU (t:15000, D:45000, s:32) F_p: prime field

Research leads/Future work

- Multi-Word with Karatsuba

$$C = 2^t \cdot A_h B_h + 2^{t/2} \cdot [(A_h - A_l) \cdot (B_l - B_h) + A_h B_h + A_l B_l] + A_l B_l$$

- Small precisions are far more efficient than big ones on GPU
  Double precision 64 times slower than single precision

Precision	Performances (TFlops)
fp32	37.42
fp64	0.585

$\Rightarrow$ Change precision in the Multi Words algorithms.

- Sequential to Asynchronous GPU calls to DGEMM
- Strassen-Winograd on top of DGEMM similar as in FFLAS ?
- Double right matrix size in MultiWord

Objective: Integrate GPU matrix multiplication in MSOLVE

Research leads/Future work

- Multi-Word with Karatsuba

$$C = 2^t \cdot A_h B_h + 2^{t/2} \cdot [(A_h - A_l) \cdot (B_l - B_h) + A_h B_h + A_l B_l] + A_l B_l$$

- Small precisions are far more efficient than big ones on GPU
  Double precision 64 times slower than single precision

Precision	Performances (TFlops)
fp32	37.42
fp64	0.585

$\Rightarrow$ Change precision in the Multi Words algorithms.

- Sequential to Asynchronous GPU calls to DGEMM
- Strassen-Winograd on top of DGEMM similar as in FFLAS ?
- Double right matrix size in MultiWord

Objective: Integrate GPU matrix multiplication in MSOLVE

Thanks!

What? Exact Polynomial System Solving → Computer Algebra, Gröbner bases
Matrix Multiplication over Prime Fields on GPU

Challenges? Computation over Finite Fields
Dense Matrix Products with large matrices
Memory limitation on GPU

Leads? MultiWord: Karatsuba / Tom-Cook, smaller precisions
Sequential to Asynchronous GPU calls to DGEMM
Strassen-Winograd on top of DGEMM like in FFLAS ?
Double right matrix size in MultiWord