FFT-based Dense Polynomial Arithmetic on Multi-cores

Yuzhen Xie
Computer Science and Artificial Intelligence Laboratory, MIT
and
Marc Moreno Maza
Ontario Research Centre for Computer Algebra, UWO

ACA 2009, Montréal, June 26

Overview (1/2)

- Developing basic polynomial algebra subroutines (BPAS) in support of polynomial system solvers and targeting hardware acceleration technologies (multi-cores, GPU, . . .)

Overview (1/2)

- Developing basic polynomial algebra subroutines (BPAS) in support of polynomial system solvers and targeting hardware acceleration technologies (multi-cores, GPU, . . .)
- We focus on dense polynomial arithmetic over finite fields, and therefore on FFT-based arithmetic.

Overview (1/2)

- Developing basic polynomial algebra subroutines (BPAS) in support of polynomial system solvers and targeting hardware acceleration technologies (multi-cores, GPU, . . .)
- We focus on dense polynomial arithmetic over finite fields, and therefore on FFT-based arithmetic.
- We have identified BALANCED BIVARIATE MULTIPLICATION as a GOOD KERNEL for dense multivariate and univariate multiplication w.r.t. parallelism and cache complexity.

Overview (1/2)

- Developing basic polynomial algebra subroutines (BPAS) in support of polynomial system solvers and targeting hardware acceleration technologies (multi-cores, GPU, ...)
- We focus on dense polynomial arithmetic over finite fields, and therefore on FFT-based arithmetic.
- We have identified BALANCED BIVARIATE MULTIPLICATION as a GOOD KERNEL for dense multivariate and univariate multiplication w.r.t. parallelism and cache complexity.
- We have developed techniques (contraction, extension, contraction + extension) to efficiently reduce to balanced bivariate multiplication.

Overview (2/2)

- BPAS ring: $\mathbb{Z}/p\mathbb{Z}[x_1, \ldots, x_n]$

Overview (2/2)

- BPAS ring: $\mathbb{Z}/p\mathbb{Z}[x_1, \ldots, x_n]$
- BPAS operations: addition, multiplication, exact division, normal form computation w.r.t. a reduced monic triangular set.

Overview (2/2)

- BPAS ring: $\mathbb{Z}/p\mathbb{Z}[x_1, \ldots, x_n]$
- BPAS operations: addition, multiplication, exact division, normal form computation w.r.t. a reduced monic triangular set.
- multiplication and normal form cover all implementation challenges.

Overview (2/2)

- BPAS ring: $\mathbb{Z}/p\mathbb{Z}[x_1, \ldots, x_n]$
- BPAS operations: addition, multiplication, exact division, normal form computation w.r.t. a reduced monic triangular set.
- multiplication and normal form cover all implementation challenges.
- BPAS assumption: 1-D FFTs are computed by a black box program which could be non-parallel.

Overview (2/2)

- BPAS ring: $\mathbb{Z}/p\mathbb{Z}[x_1, \ldots, x_n]$
- BPAS operations: addition, multiplication, exact division, normal form computation w.r.t. a reduced monic triangular set.
- multiplication and normal form cover all implementation challenges.
- BPAS assumption: 1-D FFTs are computed by a black box program which could be non-parallel.
- We rely on the `modpn` C library for serial 1-D FFTs.

Overview (2/2)

- BPAS ring: $\mathbb{Z}/p\mathbb{Z}[x_1, \ldots, x_n]$
- BPAS operations: addition, multiplication, exact division, normal form computation w.r.t. a reduced monic triangular set.
- multiplication and normal form cover all implementation challenges.
- BPAS assumption: 1-D FFTs are computed by a black box program which could be non-parallel.
- We rely on the `modpn` C library for serial 1-D FFTs.
- We use the multi-threaded programming model of (M. Frigo, C.E. Leiserson, K. H. Randall, 1998) and cache model of (M. Frigo, C.E. Leiserson, H. Prokop, S. Ramachandra 1999)
- Our concurrency platform is Cilk++, see `http://www.fftw.org/`

Outline

- Performance evaluation of FFT- vs TFT-based balanced bivariate multiplication.
- Optimizing our kernel (balanced bivariate multiplication):

(S. Huss-Lederman, E. H. Jacobson, A. Tsao, T. Turnbull, J. R. Johnson, 1996)

Outline

- Performance evaluation of FFT- vs TFT-based balanced bivariate multiplication.
- Optimizing our kernel (balanced bivariate multiplication):
  - Determining cut-off criteria between the different algorithms and implementations;

(S. Huss-Lederman, E. H. Jacobson, A. Tsao, T. Turnbull, J. R. Johnson, 1996)

Outline

- Performance evaluation of FFT- vs TFT-based balanced bivariate multiplication.
- Optimizing our kernel (balanced bivariate multiplication):
  - Determining cut-off criteria between the different algorithms and implementations;
  - Starting with theoretical analysis to narrow the solutions; and

(S. Huss-Lederman, E. H. Jacobson, A. Tsao, T. Turnbull, J. R. Johnson, 1996)

Outline

- Performance evaluation of FFT- vs TFT-based balanced bivariate multiplication.
- Optimizing our kernel (balanced bivariate multiplication):
  - Determining cut-off criteria between the different algorithms and implementations;
  - Starting with theoretical analysis to narrow the solutions; and
  - Finalizing with experimental study.

(S. Huss-Lederman, E. H. Jacobson, A. Tsao, T. Turnbull, J. R. Johnson, 1996)

Outline

- Performance evaluation of FFT- vs TFT-based balanced bivariate multiplication.
- Optimizing our kernel (balanced bivariate multiplication):
  - Determining cut-off criteria between the different algorithms and implementations;
  - Starting with theoretical analysis to narrow the solutions; and
  - Finalizing with experimental study.

  (S. Huss-Lederman, E. H. Jacobson, A. Tsao, T. Turnbull, J. R. Johnson, 1996)
- Obtaining efficient parallel computation of normal forms:

Outline

- Performance evaluation of FFT- vs TFT-based balanced bivariate multiplication.
- Optimizing our kernel (balanced bivariate multiplication):
  - Determining cut-off criteria between the different algorithms and implementations;
  - Starting with theoretical analysis to narrow the solutions; and
  - Finalizing with experimental study.

  (S. Huss-Lederman, E. H. Jacobson, A. Tsao, T. Turnbull, J. R. Johnson, 1996)
- Obtaining efficient parallel computation of normal forms:
  - Combining parallel codes for multiplication and normal forms;

Outline

- Performance evaluation of FFT- vs TFT-based balanced bivariate multiplication.
- Optimizing our kernel (balanced bivariate multiplication):
  - Determining cut-off criteria between the different algorithms and implementations;
  - Starting with theoretical analysis to narrow the solutions; and
  - Finalizing with experimental study.

  (S. Huss-Lederman, E. H. Jacobson, A. Tsao, T. Turnbull, J. R. Johnson, 1996)
- Obtaining efficient parallel computation of normal forms:
  - Combining parallel codes for multiplication and normal forms;
  - Estimating the expected parallelism and;

Outline

- Performance evaluation of FFT- vs TFT-based balanced bivariate multiplication.

- Optimizing our kernel (balanced bivariate multiplication):
  - Determining cut-off criteria between the different algorithms and implementations;
  - Starting with theoretical analysis to narrow the solutions; and
  - Finalizing with experimental study.

  (S. Huss-Lederman, E. H. Jacobson, A. Tsao, T. Turnbull, J. R. Johnson, 1996)

- Obtaining efficient parallel computation of normal forms:
  - Combining parallel codes for multiplication and normal forms;
  - Estimating the expected parallelism and;
  - Measuring the actual speed-up for various degree patterns of practical interest.

FFT-based Multivariate Multiplication (Recall)

- Let $\mathbf{k}$ be a finite field and f, $g \in \mathbf{k}[x_1 < \cdots < x_n]$ be polynomials with $n \geq 2$.
- Define $d_i = \deg(f, x_i)$ and $d'_i = \deg(g, x_i)$, for all i.
- Assume there exists a primitive s_i-th root unity $\omega_i \in \mathbf{k}$, for all i, where s_i is a power of 2 satisfying $s_i \geq d_i + d'_i + 1$.

FFT-based Multivariate Multiplication (Recall)

- Let $\mathbf{k}$ be a finite field and f, $g \in \mathbf{k}[x_1 < \cdots < x_n]$ be polynomials with $n \geq 2$.
- Define $d_i = \deg(f, x_i)$ and $d'_i = \deg(g, x_i)$, for all i.
- Assume there exists a primitive s_i-th root unity $\omega_i \in \mathbf{k}$, for all i, where s_i is a power of 2 satisfying $s_i \geq d_i + d'_i + 1$.

Then fg can be computed as follows.

Step 1. Evaluate f and g at each point P (i.e. $f(P), g(P)$) of the n-dimensional grid $((\omega_1^{e_1}, \ldots, \omega_n^{e_n}), 0 \leq e_1 < s_1, \ldots, 0 \leq e_n < s_n)$ via n-D FFT.

FFT-based Multivariate Multiplication (Recall)

- Let $\mathbf{k}$ be a finite field and f, $g \in \mathbf{k}[x_1 < \cdots < x_n]$ be polynomials with $n \geq 2$.
- Define $d_i = \deg(f, x_i)$ and $d'_i = \deg(g, x_i)$, for all i.
- Assume there exists a primitive s_i-th root unity $\omega_i \in \mathbf{k}$, for all i, where s_i is a power of 2 satisfying $s_i \geq d_i + d'_i + 1$.

Then fg can be computed as follows.

Step 1. Evaluate f and g at each point P (i.e. $f(P), g(P)$) of the n-dimensional grid $((\omega_1^{e_1}, \ldots, \omega_n^{e_n}), 0 \leq e_1 < s_1, \ldots, 0 \leq e_n < s_n)$ via n-D FFT.

Step 2. Evaluate fg at each point P of the grid, simply by computing $f(P)g(P)$,

FFT-based Multivariate Multiplication (Recall)

- Let $\mathbf{k}$ be a finite field and f, $g \in \mathbf{k}[x_1 < \cdots < x_n]$ be polynomials with $n \geq 2$.
- Define $d_i = \deg(f, x_i)$ and $d'_i = \deg(g, x_i)$, for all i.
- Assume there exists a primitive s_i-th root unity $\omega_i \in \mathbf{k}$, for all i, where s_i is a power of 2 satisfying $s_i \geq d_i + d'_i + 1$.

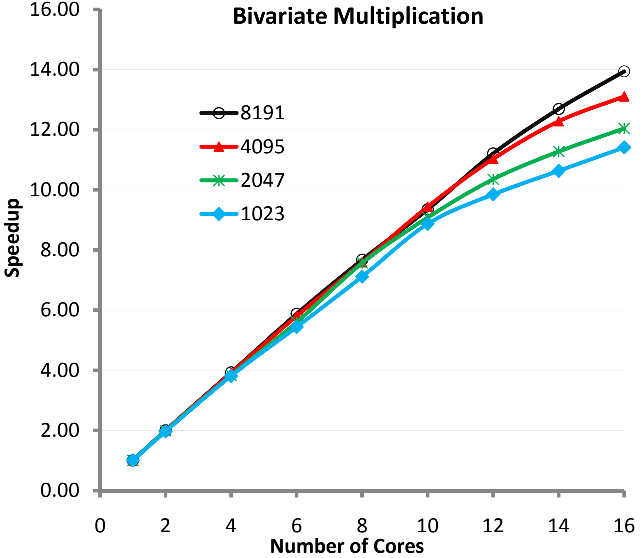
Then fg can be computed as follows.

Step 1. Evaluate f and g at each point P (i.e. $f(P), g(P)$) of the n-dimensional grid $((\omega_1^{e_1}, \ldots, \omega_n^{e_n}), 0 \leq e_1 < s_1, \ldots, 0 \leq e_n < s_n)$ via n-D FFT.

Step 2. Evaluate fg at each point P of the grid, simply by computing $f(P)g(P)$,

Step 3. Interpolate fg (from its values on the grid) via n-D FFT.

Balanced Bivariate Multiplication $(d_1 = d_2 = d'_1 = d'_2)$



Balanced Bivariate Multiplication

Table: Performance evaluation by VTune for TFT- and FFT-based mult.

	d_1 d_2	Inst. Ret. ($\times 10^9$)	Clocks per Inst. Ret. (CPI)	L2 Cache Miss Rate ($\times 10^{-3}$)	Modif. Data Shar. Ratio ($\times 10^{-3}$)	Time on 8 Cores (s)
TFT	2047 2047	44	0.794	0.423	0.215	0.86
	2048 2048	52	0.752	0.364	0.163	1.01
	2047 4095	89	0.871	0.687	0.181	2.14
	2048 4096	106	0.822	0.574	0.136	2.49
	4095 4095	179	0.781	0.359	0.141	3.72
	4096 4096	217	0.752	0.309	0.115	4.35
FFT	2047 2047	38	0.751	0.448	0.106	0.74
	2048 2048	145	0.652	0.378	0.073	2.87
	2047 4095	79	0.849	0.745	0.122	1.94
	2048 4096	305	0.765	0.698	0.094	7.64
	4095 4095	160	0.751	0.418	0.074	3.15
	4096 4096	622	0.665	0.353	0.060	12.42

Balanced Bivariate Multiplication

Table: Performance eval. by Cilkscreen for TFT- and FFT-based mult.

	d_1 d_2	Span/ Burdened Span ($\times 10^9$)	Parallelism/ Burdened Parallelism	Speedup Estimate 4P	8P	16P
TFT	2047 2047	0.613/0.614	74.18/74.02	3.69-4	6.77-8	11.63-16
	2048 2048	0.615/0.616	86.35/86.17	3.74-4	6.96-8	12.22-16
	2047 4095	0.118/0.118	92.69/92.58	3.79-4	7.09-8	12.54-16
	2048 4096	1.184/1.185	105.41/105.27	3.80-4	7.19-8	12.88-16
	4095 4095	2.431/2.433	79.29/79.24	3.71-4	6.86-8	11.89-16
	4096 4096	2.436/2.437	91.68/91.63	3.76-4	7.03-8	12.43-16
FFT	2047 2047	0.612/0.613	65.05/64.92	3.64-4	6.59-8	11.08-16
	2048 2048	0.619/0.620	250.91/250.39	3.80-4	7.50-8	14.55-16
	2047 4095	1.179/1.180	82.82/82.72	3.77-4	6.99-8	12.23-16
	2048 4096	1.190/1.191	321.75/321.34	3.80-4	7.60-8	14.82-16
	4095 4095	2.429/2.431	69.39/69.35	3.66-4	6.68-8	11.35-16
	4096 4096	2.355/2.356	166.30/166.19	3.80-4	7.47-8	13.87-16

Cut-off Criteria Estimates

- Balanced input: $d_1 + {d'}_1 \simeq d_2 + {d'}_2$.
- Moreover d_i and ${d'}_i$ are quite close, for all i.
- Consequently we assume $d := d_1 = {d'}_1 = d_2 = {d'}_2$ with $\in [2^k, 2^{k-1})$.
- We have developed a MAPLE package for polynomials in $\mathbb{Q}[k, 2^k]$ targeting complexity analysis.

Cut-off Criteria Estimates

For $d \in [2^k, 2^{k-1})$ the work of FFT-based bivariate multiplication is $48 \times 4^k(3k+7)$.

Table: Work estimates of TFT-based bivariate multiplication

d	Work
2^k	$3(2^{k+1}+1)^2(7+3k)$
$2^k + 2^{k-1}$	$81\ 4^k k + 270\ 4^k + 54\ 2^k k + 180\ 2^k + 9k + 30$
$2^k + 2^{k-1} + 2^{k-2}$	$\frac{441}{4}\ 4^k k + \frac{735}{2}\ 4^k + 63\ 2^k k + 210\ 2^k + 9k + 30$
$2^k + 2^{k-1} + 2^{k-2} + 2^{k-3}$	$\frac{2025}{16}\ 4^k k + \frac{3375}{2}\ 4^k + \frac{135}{2}\ 2^k k + 225\ 2^k + 9k + 30$

Cut-off Criteria Estimates

$d := 2^k + c_1 2^{k-1} + \cdots + c_7 2^{k-7}$ where each $c_1, \ldots, c_7 \in \{0, 1\}$.

Table: Degree cut-off estimate

$(c_1, c_2, c_3, c_4, c_5, c_6, c_7)$	Range for which this is a cut-off
$(1,1,1,0,0,0,0)$	$3 \leq k \leq 5$
$(1,1,1,0,1,0,0)$	$5 \leq k \leq 7$
$(1,1,1,0,1,1,0)$	$6 \leq k \leq 9$
$(1,1,1,0,1,1,1)$	$7 \leq k \leq 11$
$(1,1,1,1,0,0,0)$	$11 \leq k \leq 13$
$(1,1,1,1,0,1,0)$	$14 \leq k \leq 18$
$(1,1,1,1,1,0,0)$	$19 \leq k \leq 28$

These results suggest that for every range $[2^k, 2^{k-1})$ that occur in practice a sharp (or minimal) degree cut-off is around $2^k + 2^{k-1} + 2^{k-2} + 2^{k-3}$.

Cut-off Criteria Measurements

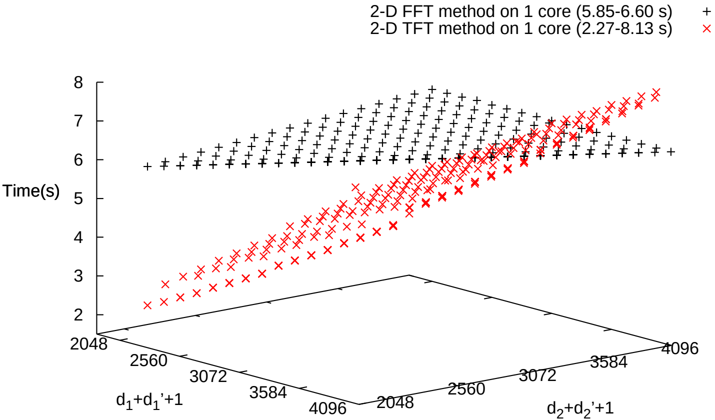


Figure: Timing of bivariate multiplication for input degree range of $[1024, 2048)$ on 1 core.

Cut-off Criteria Measurements

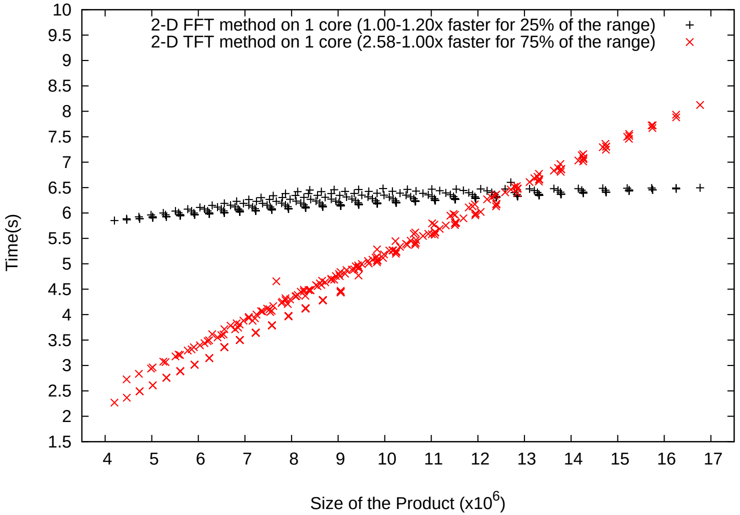


Figure: Size cut-off for input degree range of $[1024, 2048)$ on 1 core.

Cut-off Criteria Measurements

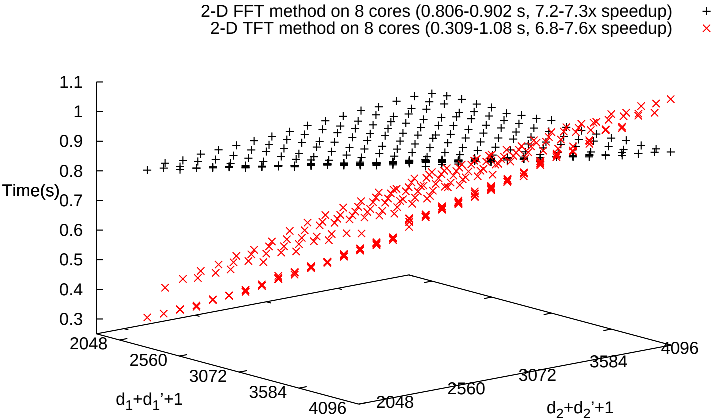


Figure: Timing of bivariate multiplication for input degree range of $[1024, 2048)$ on 8 cores.

Cut-off Criteria Measurements

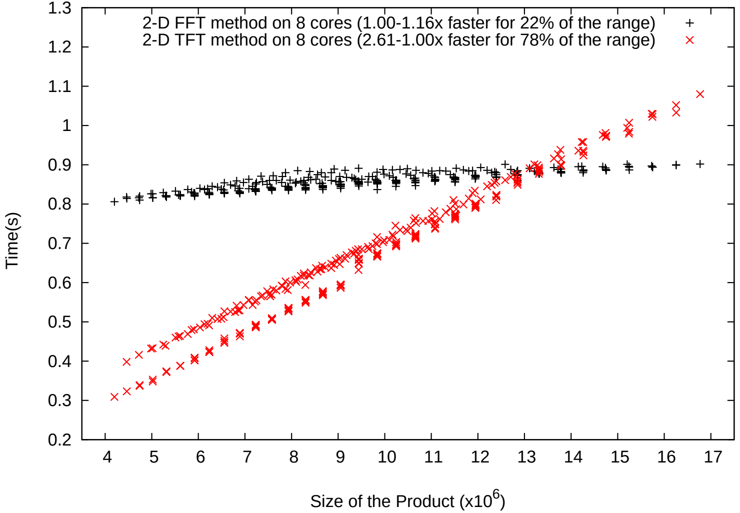


Figure: Size cut-off for input degree range of $[1024, 2048)$ on 8 cores.

Cut-off Criteria Measurements

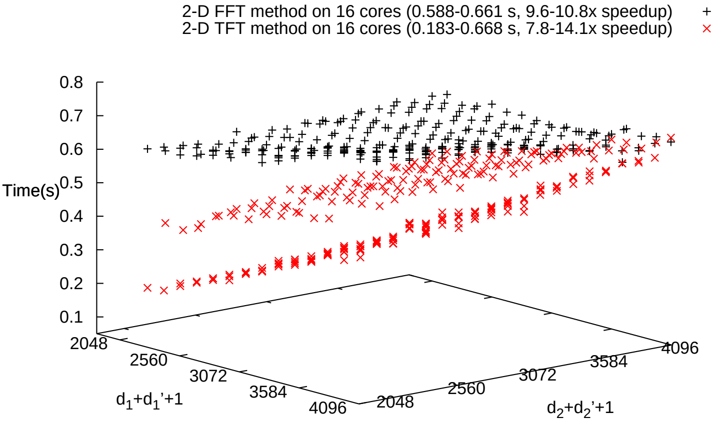


Figure: Timing of bivariate multiplication for input degree range of $[1024, 2048)$ on 16 cores.

Cut-off Criteria Measurements

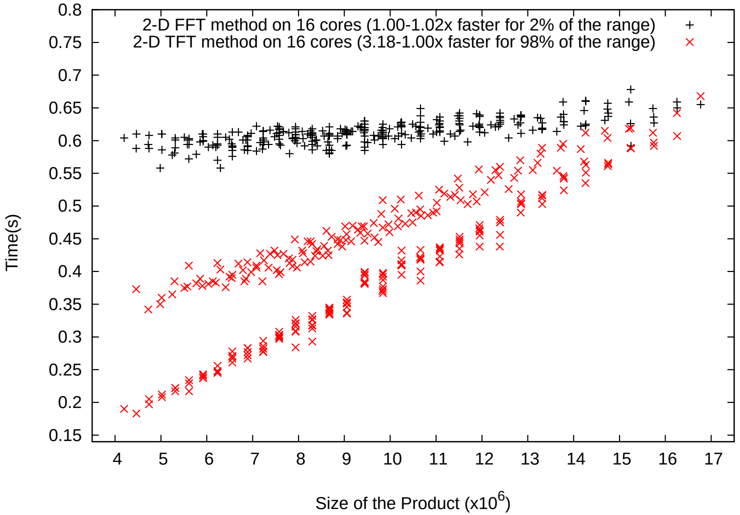


Figure: Size cut-off for input degree range of $[1024, 2048)$ on 16 cores.

Parallel Computation of Normal Forms

- In symbolic computation, normal form computations are used for simplification and equality test of algebraic expressions modulo a set of relations.

$$y^3x + yx^2 \equiv 1 - y \mod x^2 + 1, y^3 + x$$

- Many algorithms (computations with algebraic numbers, Gröbner basis computation) involve intensively normal form computations.
- We rely on an algorithm (Li, Moreno Maza and Schost 2007) which extends the fast division trick (Cook 66) (Sieveking 72) (Kung 74).
- The main idea is to efficiently reduce division to multiplication (via power series inversion).
- Preliminary attemp of parallelizing this algorithm (Li, Moreno Maza, 1997) reached a limited success.

Parallel Computation of Normal Forms

$\text{NormalForm}_1(f, \{g_1\} \subset \mathbf{k}[x_1])$

$S_1 := \mathrm{Rev}(g_1)^{-1} \mod x_1^{\deg(f,x_1)-\deg(g_1,x_1)+1}$

$D := \mathrm{Rev}(A)S_1 \mod x_1^{\deg(f,x_1)-\deg(g_1,x_1)+1}$

$D := g_1 \,\mathrm{Rev}(D)$

return $A - D$

$\text{NormalForm}_i(f, \{g_1, \ldots, g_i\} \subset \mathbf{k}[x_1, \ldots, x_i])$

$A := \mathrm{map}(\text{NormalForm}_{i-1}, \mathrm{Coeffs}(f, x_i), \{g_1, \ldots, g_{i-1}\})$

$S_i := \mathrm{Rev}(g_i)^{-1} \mod g_1, \ldots, g_{i-1}, x_i^{\deg(f,x_i)-\deg(g_i,x_i)+1}$

$D := \mathrm{Rev}(A)S_i \mod x_i^{\deg(f,x_i)-\deg(g_i,x_i)+1}$

$D := \mathrm{map}(\text{NormalForm}_{i-1}, \mathrm{Coeffs}(D, x_{)}, \{g_1, \ldots, g_{i-1}\})$

$D := g_i \,\mathrm{Rev}(D)$

$D := \mathrm{map}(\text{NormalForm}_{i-1}, \mathrm{Coeffs}(D, x_i), \{g_1, \ldots, g_{i-1}\})$

return $A - D$

Parallel Computation of Normal Forms

Define $\delta_i := \deg(g_i, x_i)$ and $\ell_i = \prod_{j=1}^{j=i} \lg(\delta_j)$. Denote by $\mathsf{W}_\mathsf{M}(\underline{\delta}_i)$ and $\mathsf{S}_\mathsf{M}(\underline{\delta}_i)$ the work and span of a multiplication algorithm.

(1) Span estimate with serial multiplication:

$$\mathsf{S}_{\mathrm{NF}}(\underline{\delta}_i) = 3\,\ell_i\,\mathsf{S}_{\mathrm{NF}}(\underline{\delta}_{i-1}) + 2\,\mathsf{W}_\mathsf{M}(\underline{\delta}_i) + \ell_i.$$

(2) Span estimate with parallel multiplication

$$\mathsf{S}_{\mathrm{NF}}(\underline{\delta}_i) = 3\,\ell_i\,\mathsf{S}_{\mathrm{NF}}(\underline{\delta}_{i-1}) + 2\,\mathsf{S}_\mathsf{M}(\underline{\delta}_i) + \ell_i.$$

Parallel Computation of Normal Forms

Define $\delta_i := \deg(g_i, x_i)$ and $\ell_i = \prod_{j=1}^{j=i} \lg(\delta_j)$. Denote by $\mathsf{W}_\mathsf{M}(\underline{\delta}_i)$ and $\mathsf{S}_\mathsf{M}(\underline{\delta}_i)$ the work and span of a multiplication algorithm.

(1) Span estimate with serial multiplication:

$$\mathsf{S}_{\mathrm{NF}}(\underline{\delta}_i) = 3\,\ell_i\,\mathsf{S}_{\mathrm{NF}}(\underline{\delta}_{i-1}) + 2\,\mathsf{W}_\mathsf{M}(\underline{\delta}_i) + \ell_i.$$

(2) Span estimate with parallel multiplication

$$\mathsf{S}_{\mathrm{NF}}(\underline{\delta}_i) = 3\,\ell_i\,\mathsf{S}_{\mathrm{NF}}(\underline{\delta}_{i-1}) + 2\,\mathsf{S}_\mathsf{M}(\underline{\delta}_i) + \ell_i.$$

- Work, span and parallelism are all **exponential** in the number of variables.
- Moreover, the number of joining threads per synchronization point grows with the partial degrees of the input polynomials.

Parallel Computation of Normal Forms

Table: Span estimates of TFT-based Normal Form for $\underline{\delta}_i = (2^k, 1, \ldots, 1)$.

i	With serial multiplication	With parallel multiplication
2	$144\ k\ 2^k + 642\ 2^k + 76\ k + 321$	$72\ k\ 2^k + 144\ 2^k + 160\ k + 312$
4	$4896\ k\ 2^k + 45028\ 2^k + 2488\ k + 22514$	$1296\ k\ 2^k + 2592\ 2^k + 6304\ k + 12528$
8	$3456576\ k\ 2^k + 71229768\ 2^k + \mathrm{o}(2^k)$	$209952\ k\ 2^k + 419904\ 2^k + \mathrm{o}(2^k)$

Table: Parallelism est. of TFT-based Normal Form for $\underline{\delta}_i = (2^k, 1, \ldots, 1)$.

i	With serial multiplication	With parallel multiplication
2	$13/8 \simeq 2$	$13/4 \simeq 3$
4	$1157/272 \simeq 4$	$1157/72 \simeq 16$
8	$5462197/192032 \simeq 29$	$5462197/11664 \simeq 469$

Parallel Computation of Normal Forms

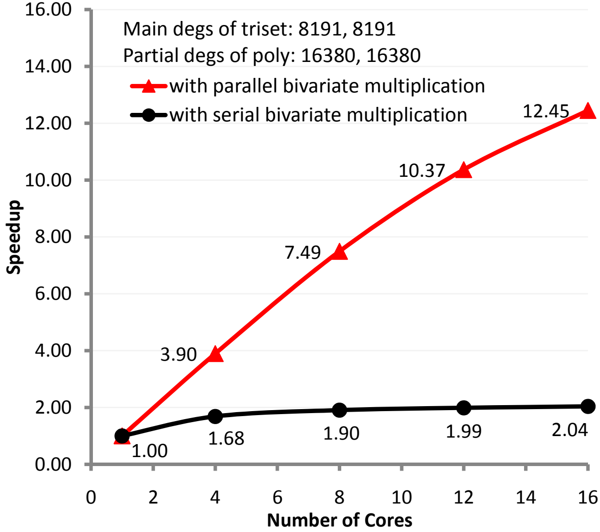


Figure: Normal form computation of a large bivariate problem.

Parallel Computation of Normal Forms

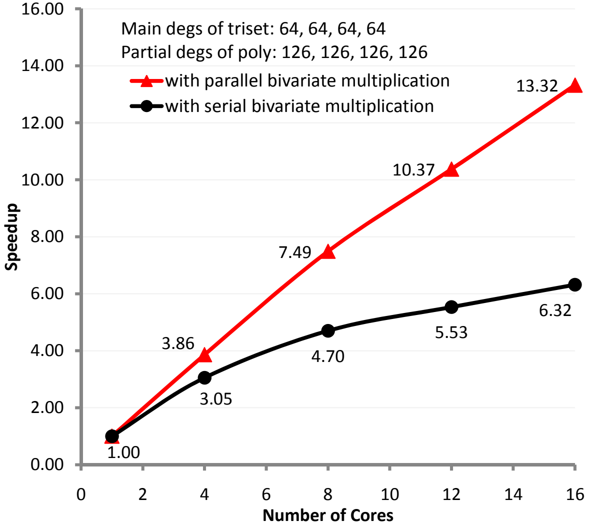


Figure: Normal form computation of a medium-sized 4-variate problem.

Parallel Computation of Normal Forms

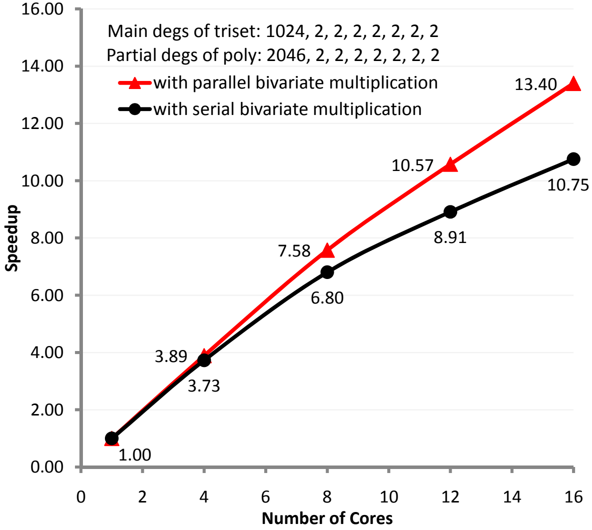


Figure: Normal form computation of an irregular 8-variate problem.

Conclusions

Summary and future work:

- We have shown that (FFT-based) balanced bivariate multiplication can be highly efficient in terms of parallelism and cache complexity.
- We have determined cut-off criteria between TFT and FFT-based for balanced bivariate multiplication.
- Not only parallel multiplication can improve the performance of parallel normal form computation,
- but also that this composition is necessary for parallel normal form computation to reach good speed-up factors on all input patterns that we have tested.

Conclusions

Summary and future work:

- We have shown that (FFT-based) balanced bivariate multiplication can be highly efficient in terms of parallelism and cache complexity.
- We have determined cut-off criteria between TFT and FFT-based for balanced bivariate multiplication.
- Not only parallel multiplication can improve the performance of parallel normal form computation,
- but also that this composition is necessary for parallel normal form computation to reach good speed-up factors on all input patterns that we have tested.

Acknowledgements:

This work was supported by NSERC and MITACS NCE of Canada, and NSF Grants 0540248, 0615215, 0541209, and 0621511. We are very grateful for the help of Professor Charles E. Leiserson at MIT, Dr. Matteo Frigo and all other members of Cilk Arts.

Thank You!