FFT-based Dense Polynomial Arithmetic on Multi-cores

Marc Moreno Maza
Ontario Research Centre for Computer Algebra (ORCCA)
University of Western Ontario, Canada
joint work with
Yuzhen Xie
SuperTech Group, MIT CSAIL

CS 4435 - CS 9624, February 1st, 2010

Introduction (1/2)

- Developing basic polynomial algebra subroutines (BPAS) in support of polynomial system solvers and targeting hardware acceleration technologies (multi-cores, GPU, ...)
- BPAS operations: $+, \times, \div$ and `normal form` computation w.r.t. a reduced monic triangular set
- `multiplication` and `normal form` cover all implementation challenges
- BPAS ring: $\mathbb{Z}/p\mathbb{Z}[x_1, \ldots, x_n]$
- We focus on dense polynomial arithmetic over finite fields, and therefore on FFT-based arithmetic.

Introduction (2/2)

- 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/TFTs, and for integer modulo arithmetic (Montgomery trick).
- We use the multi-threaded programming model of (Frigo, Leiserson and Randall, 1998) and cache model of (Frigo, Leiserson, Prokop, and Ramachandra 1999)
- Our concurrency platform is Cilk++:
  - provably efficient work-stealing scheduling
  - ease-of-use and low-overhead parallel constructs: `cilk_for`, `cilk_spawn`, `cilk_sync`
  - Cilkscreen for data race detection and parallelism analysis

Outline

(1) Identifying **Balanced Bivariate Multiplication** as a good kernel for dense multivariate and univariate multiplication w.r.t. parallelism and cache complexity

(2) Reducing to balanced bivariate multiplication by contraction, extension, and contraction+extension techniques

(3) Optimizing this kernel:
– performance evaluation by VTune and Cilkscreen
– determining cut-off criteria between the different algorithms and implementations

(4) Obtaining efficient parallel computation of normal forms by composing the parallelism of `multiplication` and that of `normal forms`

Combining theoretical analysis with experimental study!

Review of 2-D FFT

Let $f(x, y) = \sum_{i=0}^{2} g_i(x) y^i$, where $g_i(x) = \sum_{j=0}^{3} c_{ij} x^j$.

(1) FFTs along x: $g_i(\omega_1^k) = \sum_{j=0}^{3} c_{ij} \omega_1^{kj}$, where $0 \leq k \leq 3$.

$$
\begin{array}{cccccccccc}
c_{00} & c_{01} & c_{02} & c_{03} & \Longrightarrow & g_0(\omega_1^0) & g_0(\omega_1^1) & g_0(\omega_1^2) & g_0(\omega_1^3) \\
c_{10} & c_{11} & c_{12} & c_{13} & \Longrightarrow & g_1(\omega_1^0) & g_1(\omega_1^1) & g_1(\omega_1^2) & g_1(\omega_1^3) \\
c_{20} & c_{21} & c_{22} & c_{23} & \Longrightarrow & g_2(\omega_1^0) & g_2(\omega_1^1) & g_2(\omega_1^2) & g_2(\omega_1^3)
\end{array}
$$

Review of 2-D FFT

Let $f(x,y) = \sum_{i=0}^{2} g_i(x) y^i$, where $g_i(x) = \sum_{j=0}^{3} c_{ij} x^j$.

(1) FFTs along x: $g_i(\omega_1^k) = \sum_{j=0}^{3} c_{ij} \omega_1^{kj}$, where $0 \le k \le 3$.

$$\begin{array}{cccccccc}
c_{00} & c_{01} & c_{02} & c_{03} & \Longrightarrow & g_0(\omega_1^0) & g_0(\omega_1^1) & g_0(\omega_1^2) & g_0(\omega_1^3) \\
c_{10} & c_{11} & c_{12} & c_{13} & \Longrightarrow & g_1(\omega_1^0) & g_1(\omega_1^1) & g_1(\omega_1^2) & g_1(\omega_1^3) \\
c_{20} & c_{21} & c_{22} & c_{23} & \Longrightarrow & g_2(\omega_1^0) & g_2(\omega_1^1) & g_2(\omega_1^2) & g_2(\omega_1^3)
\end{array}$$

(2) FFTs along y: $f(\omega_1^k, \omega_2^\ell) = \sum_{i=0}^{2} g_i(\omega_1^k) \omega_2^{\ell i}$, where $0 \le k \le 3$ and $0 \le \ell \le 2$.

$$\begin{array}{ccccccc}
g_0(\omega_1^0) & g_1(\omega_1^0) & g_2(\omega_1^0) & \Longrightarrow & f(\omega_1^0, \omega_2^0) & f(\omega_1^0, \omega_2^1) & f(\omega_1^0, \omega_2^2) \\
g_0(\omega_1^1) & g_1(\omega_1^1) & g_2(\omega_1^1) & \Longrightarrow & f(\omega_1^1, \omega_2^0) & f(\omega_1^1, \omega_2^1) & f(\omega_1^1, \omega_2^2) \\
g_0(\omega_1^2) & g_1(\omega_1^2) & g_2(\omega_1^2) & \Longrightarrow & f(\omega_1^2, \omega_2^0) & f(\omega_1^2, \omega_2^1) & f(\omega_1^2, \omega_2^2) \\
g_0(\omega_1^3) & g_1(\omega_1^3) & g_2(\omega_1^3) & \Longrightarrow & f(\omega_1^3, \omega_2^0) & f(\omega_1^3, \omega_2^1) & f(\omega_1^3, \omega_2^2)
\end{array}$$

Remark 1: This procedure evaluates $f(x,y)$ on the grid $(\omega_1^k, \omega_2^\ell)$, for $0 \le k \le 3$ and $0 \le \ell \le 2$.

FFT-based Multivariate Multiplication

- 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 of unity $\omega_i \in \mathbf{k}$, for all i, where s_i is a power of 2 satisfying $s_i \geq d_i + d'_i + 1$.

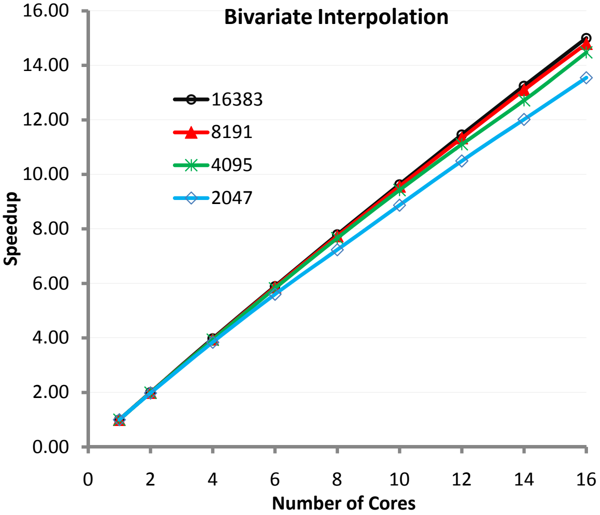
Then fg can be computed as follows.

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 $f(P)g(P)$,

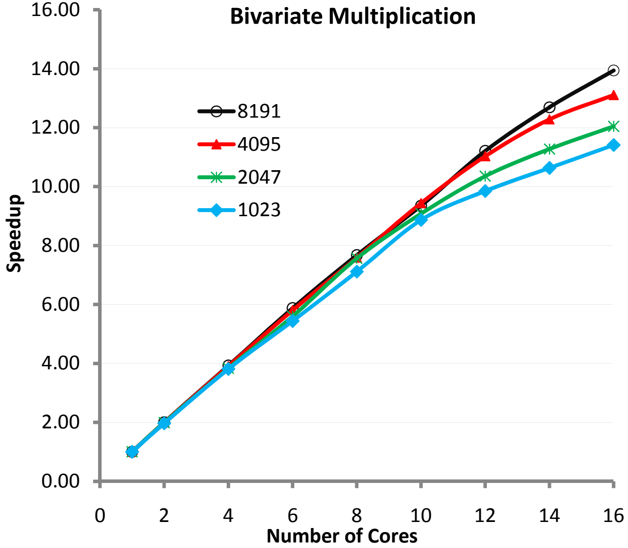
Step 3. Interpolate fg (from its values on the grid) via n-D FFT.

Performance of Bivariate Interpolation in Step 3 $(d_1 = d_2)$

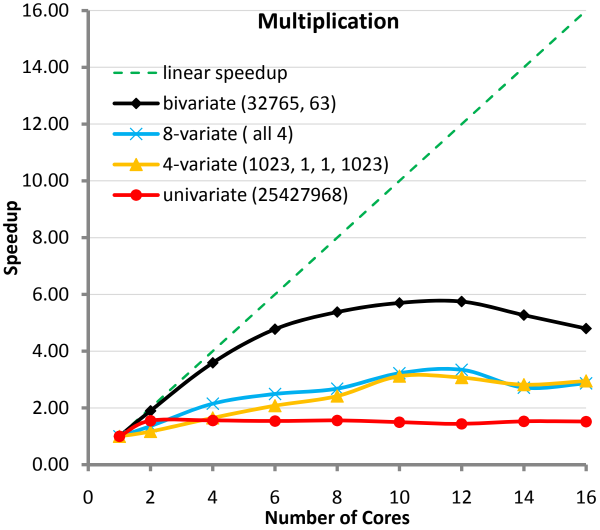


Thanks to Dr. Frigo for his cache-efficient code for matrix transposition!

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



Challenges: Irregular Input Data



These unbalanced data pattern are common in symbolic computation.

Performance Analysis by VTune

No.	Size of Two Input Polynomials	Product Size
1	8191×8191	268402689
2	259575×258	268401067
3	63×63×63×63	260144641
4	8 vars. of deg. 5	214358881

No.	INST_ RETIRED. ANY$\times 10^9$	Clocks per Instruction Retired	L2 Cache Miss Rate ($\times 10^{-3}$)	Modified Data Sharing Ratio ($\times 10^{-3}$)	Time on 8 Cores (s)
1	659.555	0.810	0.333	0.078	16.15
2	713.882	0.890	0.735	0.192	19.52
3	714.153	0.854	1.096	0.635	22.44
4	1331.340	1.418	1.177	0.576	72.99

Complexity Analysis (1/2)

- Let $s = s_1 \cdots s_n$. The number of operations in $\mathbf{k}$ for computing fg via n-D FFT is

$$\frac{9}{2}\sum_{i=1}^{n}(\prod_{j\neq i} s_j)s_i \lg(s_i) + (n+1)s = \frac{9}{2}s \lg(s) + (n+1)s.$$

- Under our 1-D FFT black box assumption, the span of *Step* 1 is

$$\tfrac{9}{2}\left(s_1 \lg(s_1) + \cdots + s_n \lg(s_n)\right),$$

and the parallelism of *Step* 1 is lower bounded by

$$s/\max(s_1, \ldots, s_n). \tag{1}$$

- Let L be the size of a cache line. For some constant $c > 0$, the number of cache misses of *Step* 1 is upper bounded by

$$n\frac{cs}{L} + cs(\frac{1}{s_1} + \cdots + \frac{1}{s_n}). \tag{2}$$

Complexity Analysis (2/2)

- Let $Q(s_1, \ldots, s_n)$ denotes the total number of cache misses for the whole algorithm, for some constant c we obtain

$$Q(s_1, \ldots, s_n) \leq cs\frac{n+1}{L} + cs(\frac{1}{s_1} + \cdots + \frac{1}{s_n}) \qquad (3)$$

- Since $\frac{n}{s^{1/n}} \leq \frac{1}{s_1} + \cdots + \frac{1}{s_n}$, we deduce

$$Q(s_1, \ldots, s_n) \leq ncs(\frac{2}{L} + \frac{1}{s^{1/n}}) \qquad (4)$$

when $s_i = s^{1/n}$ holds for all i.

Remark 2: For $n \geq 2$, Expr. (4) is minimized at $n = 2$ and $s_1 = s_2 = \sqrt{s}$. Moreover, when $n = 2$, under a fixed $s = s_1 s_2$, Expr. (1) is maximized at $s_1 = s_2 = \sqrt{s}$.

Our Solutions

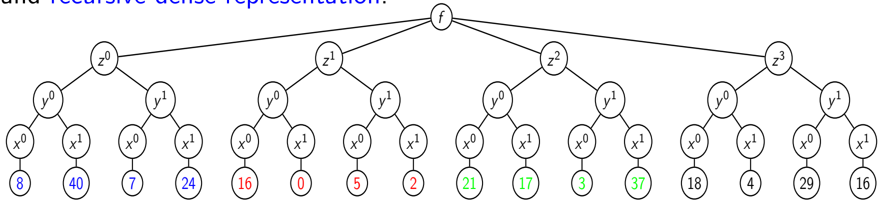
(1) Contraction to bivariate from multivariate

(2) Extension from univariate to bivariate

(3) Balanced multiplication by extension and contraction

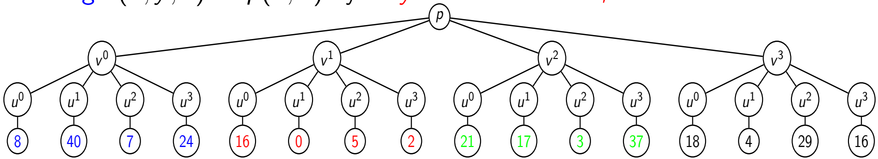
Solution 1: Contraction to Bivariate from Multivar.

Example. Let $f \in \mathbf{k}[x, y, z]$ where $\mathbf{k} = \mathbb{Z}/41\mathbb{Z}$, with $d_x = d_y = 1$, $d_z = 3$, and recursive dense representation:



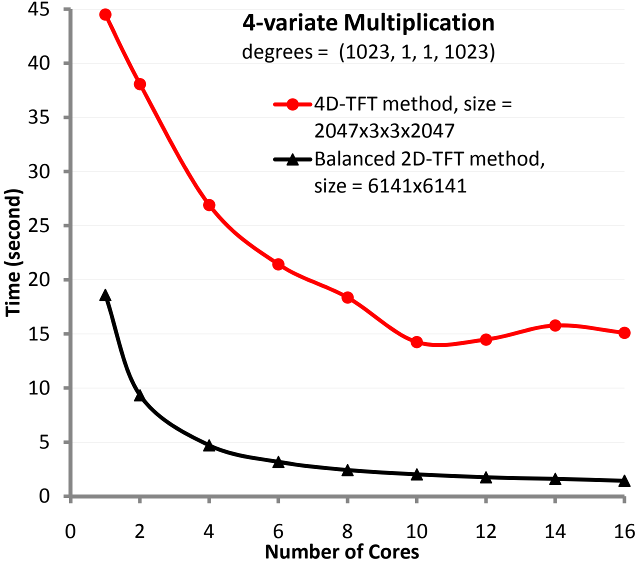
- ⋆ The coefficients (not monomials) are stored in a contiguous array.
- ⋆ The coeff. of $x^{e_1} y^{e_2} z^{e_3}$ has index $e_1 + (d_x + 1)e_2 + (d_x + 1)(d_y + 1)e_3$.

Contracting $f(x, y, z)$ to $p(u, v)$ by $x^{e_1} y^{e_2} \mapsto u^{e_1 + (d_x+1)e_2}$, $z^{e_3} \mapsto v^{e_3}$:

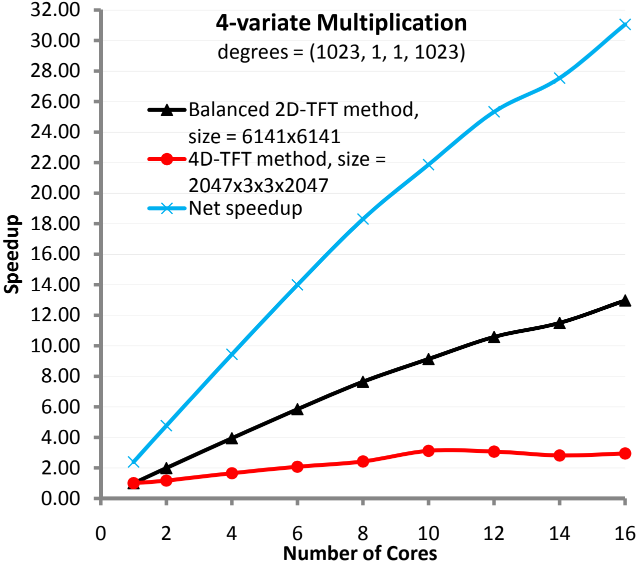


Remark 3: The coefficient array is "essentially" unchanged by contraction, which is a property of recursive dense representation.

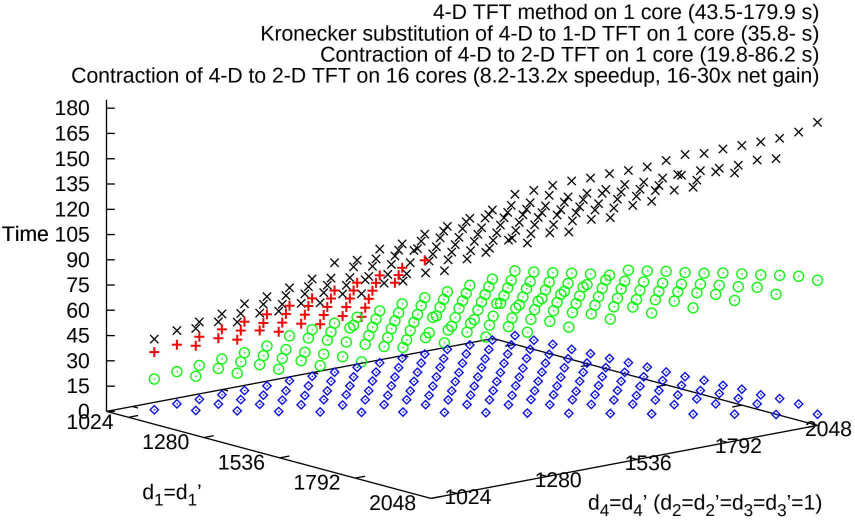
Performance of Contraction (timing)



Performance of Contraction (speedup)



Performance of Contraction for a Large Range of Problems



Solution 2: Extension from Univariate to Bivariate

Example: Consider $f, g \in \mathbf{k}[x]$ univariate, with $\deg(f) = 7$ and $\deg(g) = 8$; fg has "dense size" 16.

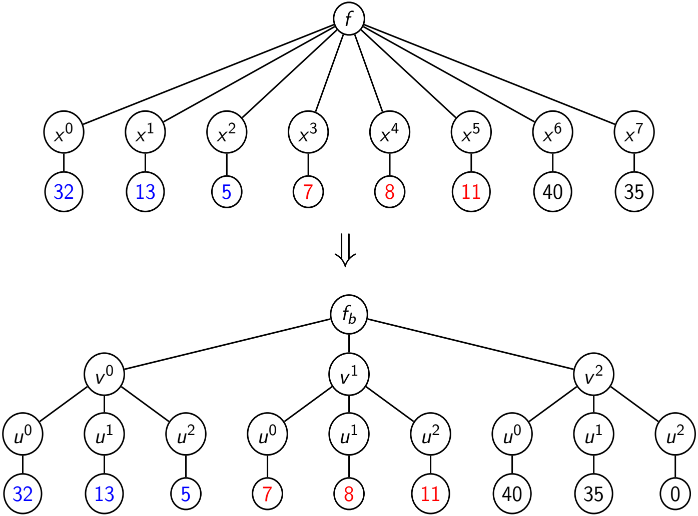
- We compute an integer b, such that fg can be performed via $f_b g_b$ using "nearly square" 2-D FFTs, where $f_b := \Phi_b(f)$, $g_b := \Phi_b(g)$ and

$$\mathbf{\Phi_b} : \ \mathbf{x^e} \longmapsto \mathbf{u}^{\mathbf{e} \operatorname{rem} \mathbf{b}} \ \mathbf{v}^{\mathbf{e} \operatorname{quo} \mathbf{b}}.$$

⋆ Here $b = 3$ works since $\deg(f_b g_b, u) = \deg(f_b g_b, v) = 4$; moreover the dense size of $f_b g_b$ is 25.

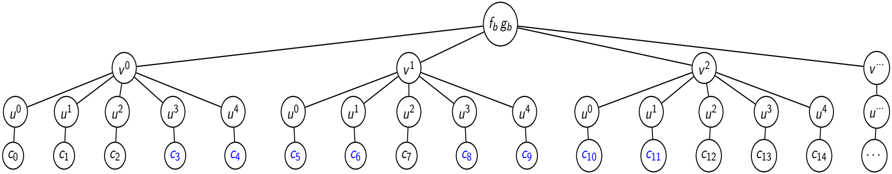
Proposition: For any non-constant $f, g \in \mathbf{k}[x]$, one can always compute b such that $|deg(f_b g_b, u) - deg(f_b g_b, v)| \leq 2$ and the dense size of $f_b g_b$ is at most twice that of fg.

Extension of $f(x)$ to $f_b(u, v)$ in Recursive Dense Representation

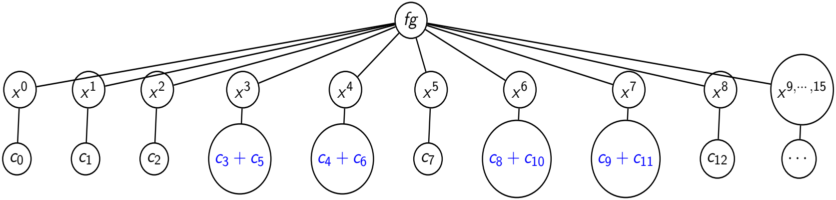


Conversion to Univariate from the Bivariate Product

- The bivariate product: $deg(f_b g_b, u) = 4, deg(f_b g_b, v) = 4.$

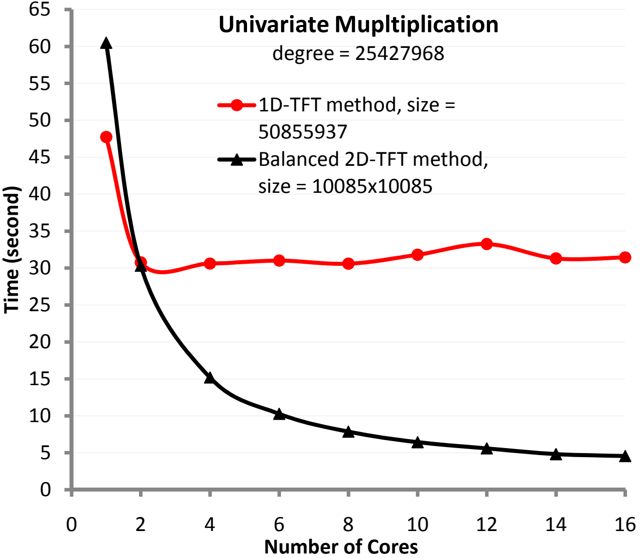


- Convert to univariate: $deg(fg, x) = 15.$

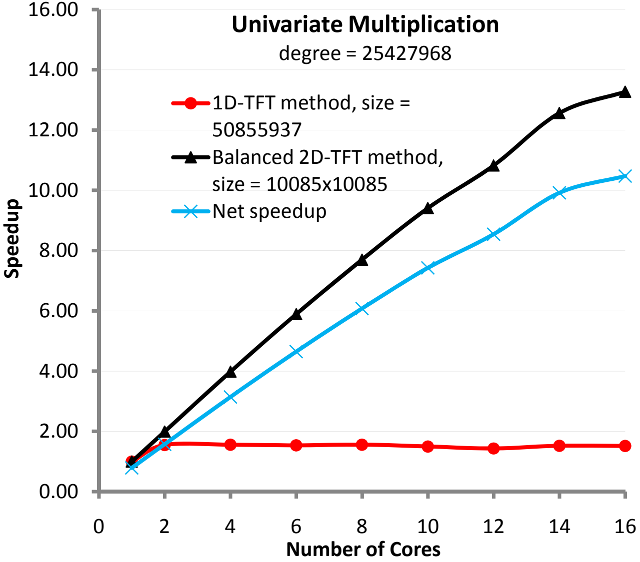


Remark 4: Converting back to fg from $f_b g_b$ requires only to traverse the coefficient array once, and perform at most $\deg(fg, x)$ additions.

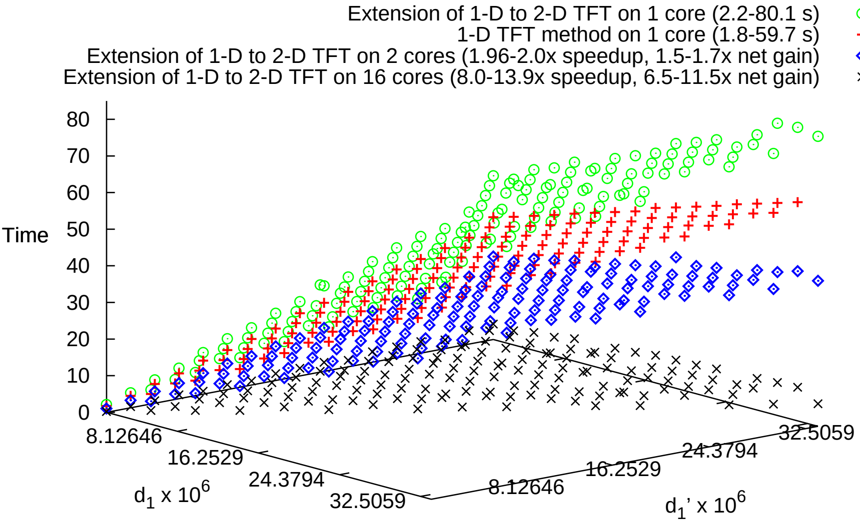
Performance of Extension (timing)



Performance of Extension (speedup)



Performance of Extension for a Large Range of Problems



Solution 3: Balanced Multiplication

Definition. A pair of bivariate polynomials $p, q \in \mathbf{k}[u, v]$ is *balanced* if $\deg(p, u) + \deg(q, u) = \deg(p, v) + \deg(q, v)$.

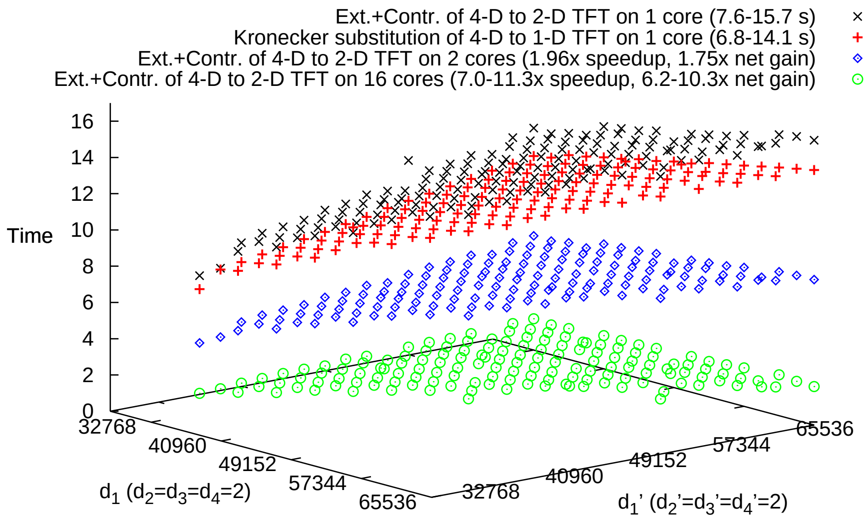
Algorithm. Let $f, g \in \mathbf{k}[x_1 < \ldots < x_n]$. W.l.o.g. one can assume $d_1 >> d_i$ and $d_1{}' >> d_i$ for $2 \leq i \leq n$ (up to variable re-ordering and contraction). Then we obtain fg by

Step 1. Extending x_1 to $\{u, v\}$.

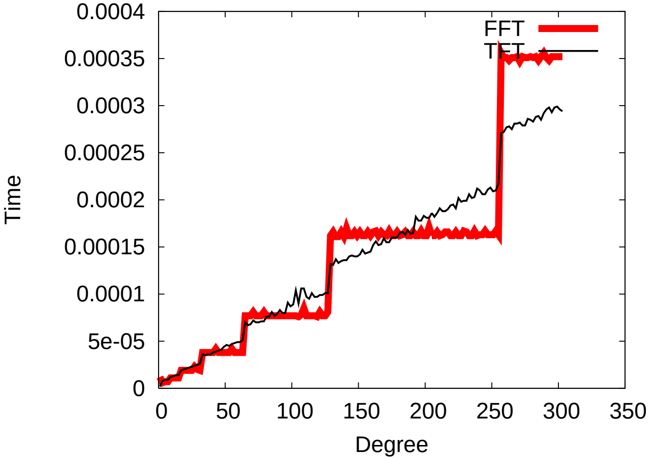
Step 2. Contracting $\{v, x_2, \ldots, x_n\}$ to v.

Remark 5: The above extension Φ_b can be determined such that f_b, g_b is (nearly) a balanced pair and $f_b g_b$ has dense size at most twice that of fg.

Performance of Balanced Mul. for a Large Range of Problems



Cut-off Criteria Estimates: TFT- vs FFT-based Methods



Performance Evaluation by VTune for TFT- and FFT-based Bivar. 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

Performance Eval. by Cilkscreen for TFT- and FFT-based Bivar. 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

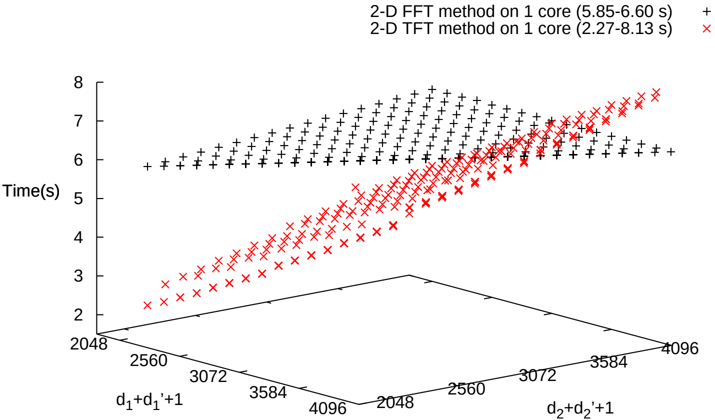


Figure: Timing of bivariate multiplication for input degree range of $[1024, 2048)$ on 1 core.

Cut-off Criteria Measurements

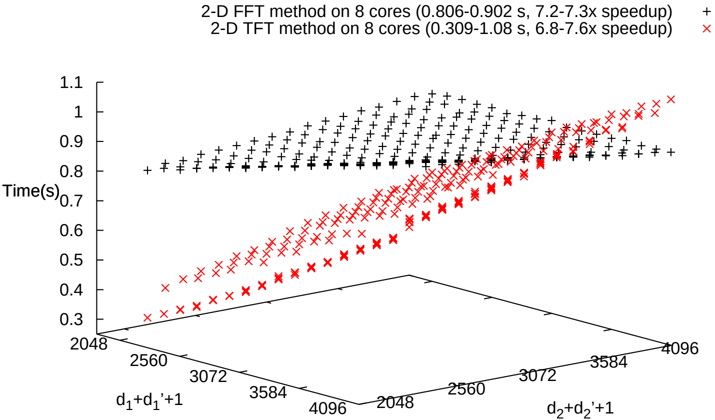


Figure: Timing of bivariate multiplication for input degree range of $[1024, 2048)$ on 8 cores.

Cut-off Criteria Measurements

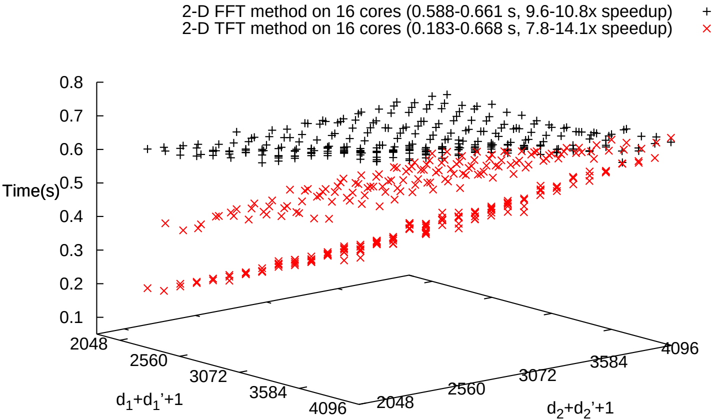


Figure: Timing of bivariate multiplication for input degree range of $[1024, 2048)$ on 16 cores.

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, 2007) 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

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

- 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

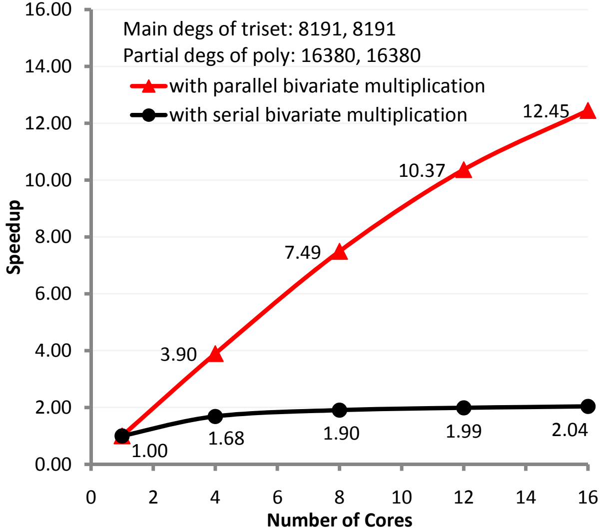


Figure: Normal form computation of a large bivariate problem.

Parallel Computation of Normal Forms

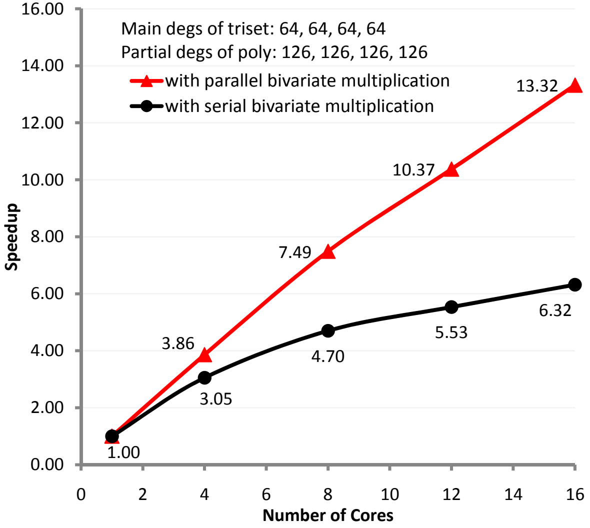


Figure: Normal form computation of a medium-sized 4-variate problem.

Parallel Computation of Normal Forms

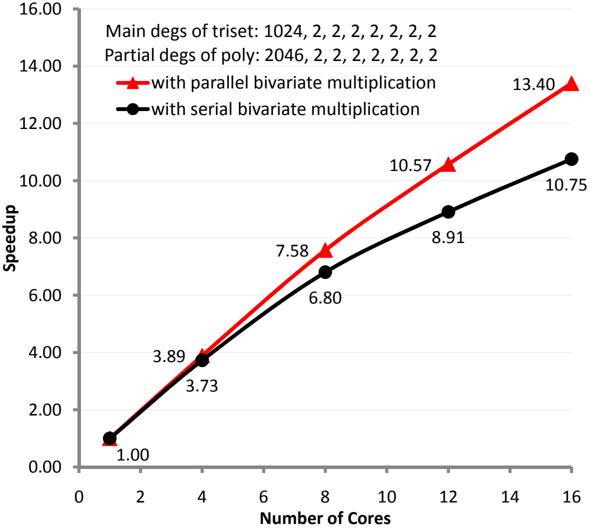


Figure: Normal form computation of an irregular 8-variate problem.

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 provided efficient techniques to reduce unbalanced input to balanced bivariate multiplication.
- Not only balanced parallel multiplication can improve the performance of parallel normal form computation, but also this composition is necessary for parallel normal form computation to reach peak performance on all input patterns that we have tested.
- Work-in-progress includes parallel resultant/GCD and a polynomial solver via triangular decompositions.

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, Dr. Matteo Frigo and all other members of SuperTech Group at MIT and Cilk Arts.

Thank You!