# Balanced Dense Polynomial Multiplication on Multicores

**Yuzhen Xie**
SuperTech Group, CSAIL MIT
joint work with
**Marc Moreno Maza**
ORCCA, UWO

ACA09, Montreal, June 26, 2009

# Introduction

**Motivation:** Multicore-enabling parallel polynomial arithmetic in computer algebra

- Fast dense polynomial multiplication via FFT
- Multivariate polynomials over finite fields

# Introduction

**Motivation:** Multicore-enabling parallel polynomial arithmetic in computer algebra

- Fast dense polynomial multiplication via FFT
- Multivariate polynomials over finite fields

**Framework:**

- Assume 1-D FFT (in particular 1-D TFT) as a black box
- Rely on the `modpn` C library (shipped with Maple 13):
  – for 1-D FFT and 1-D TFT computations,
  – for integer modulo arithmetic (Montgomery trick)
- Implement in Cilk++ targeting multi-cores:
  – 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

# FFT-based Multivariate Multiplication
## (Algorithm Review)

- Let $\mathbf{k}$ be a field and $f, g \in \mathbf{k}[x_1 < \cdots < x_n]$ be polynomials.
- 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
## (Algorithm Review)

- Let $\mathbf{k}$ be a field and $f$, $g \in \mathbf{k}[x_1 < \cdots < x_n]$ be polynomials.
- 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
## (Algorithm Review)

- Let $\mathbf{k}$ be a field and $f$, $g \in \mathbf{k}[x_1 < \cdots < x_n]$ be polynomials.
- 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 (Algorithm Review)

- Let $\mathbf{k}$ be a field and $f$, $g \in \mathbf{k}[x_1 < \cdots < x_n]$ be polynomials.
- 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 $s_i \geq d_i + d'_i + 1$.

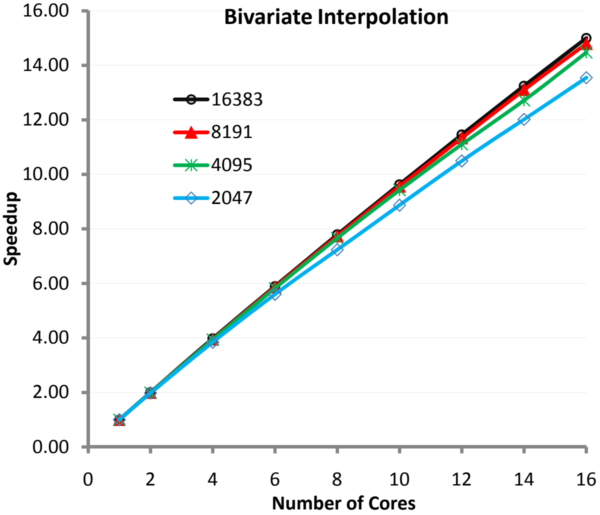
Then $fg$ can be computed as follows.

*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 $f(P)g(P)$,

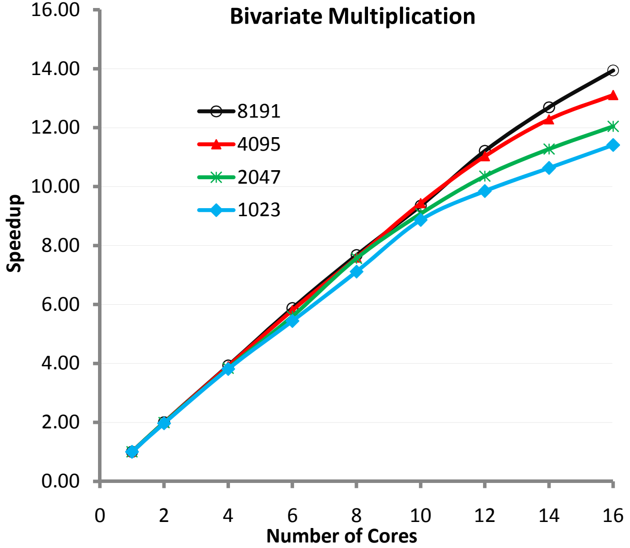
*Step* 3. Interpolate $fg$ (from its values on the grid) via $n$-D FFT.

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

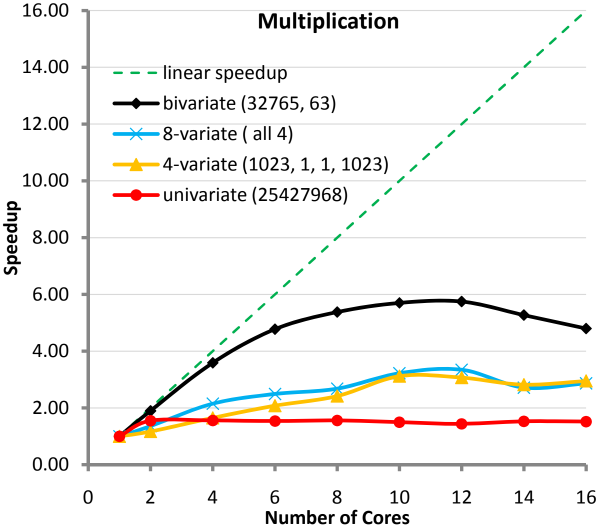



Thanks to Dr. Frigo for his cache-efficient code for matrix transposition!

# 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×$10^9$ | 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

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

# 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

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

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

# 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

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

# 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

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

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

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

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

# 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

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

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

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

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

**Remark 1:** 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

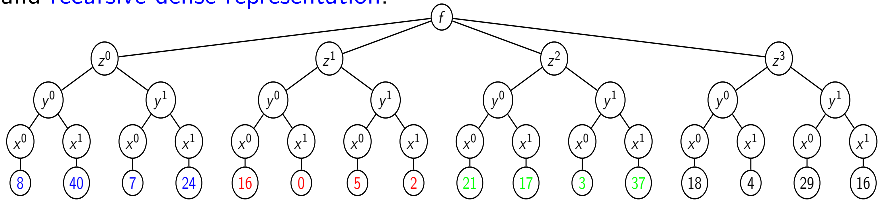
(1) Contraction to bivariate from multivariate

(2) Extension from univariate to bivariate

(3) Balanced multiplication by extension and contraction

# Solution 1: Contraction to Bivariate from Multivar.

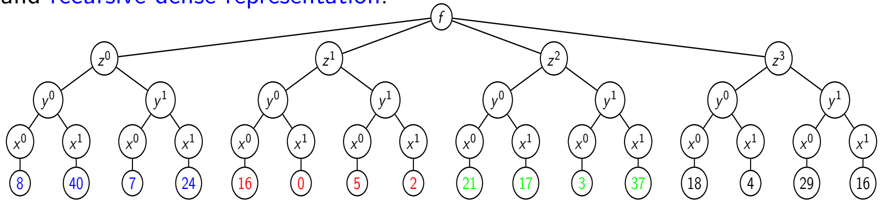
**Example.** Let $f \in \mathbf{k}[x, y, z]$ where $\mathbf{k} = \mathbb{Z}/41\mathbb{Z}$, with $d_x = d_y = 1$, $d_z = 3$, and recursive dense representation:




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

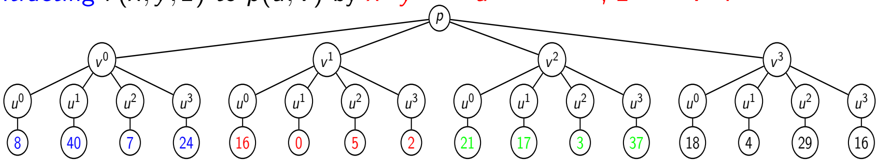
# Solution 1: Contraction to Bivariate from Multivar.

**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:



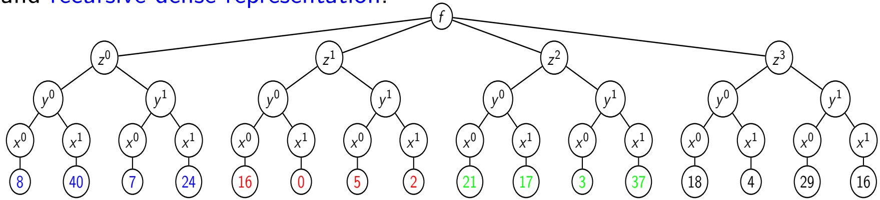

- ⋆ The coefficients (not monomials) are stored in a contiguous array.
- ⋆ Index monomial $x^{e_1} y^{e_2} z^{e_3}$ by $e_1 + (d_x + 1)e_2 + (d_x + 1)(d_y + 1)e_3$.

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



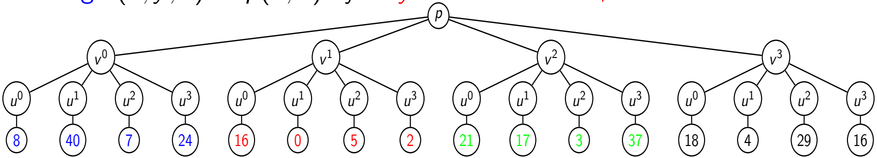
# Solution 1: Contraction to Bivariate from Multivar.

**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:



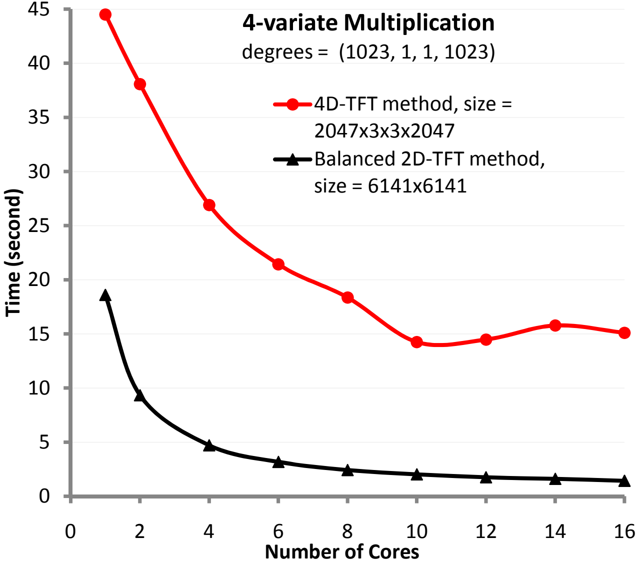

- ⋆ The coefficients (not monomials) are stored in a contiguous array.
- ⋆ Index monomial $x^{e_1} y^{e_2} z^{e_3}$ by $e_1 + (d_x + 1)e_2 + (d_x + 1)(d_y + 1)e_3$.

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

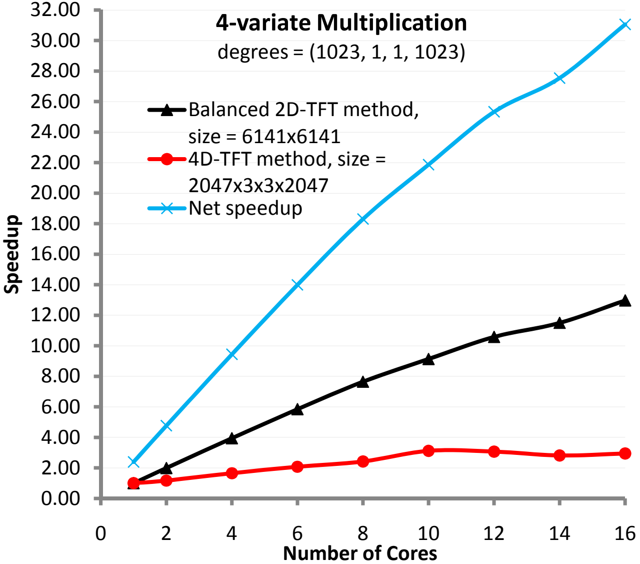



**Remark 2**: The coefficient array is "essentially" unchanged by contraction, which is a property of recursive dense representation.

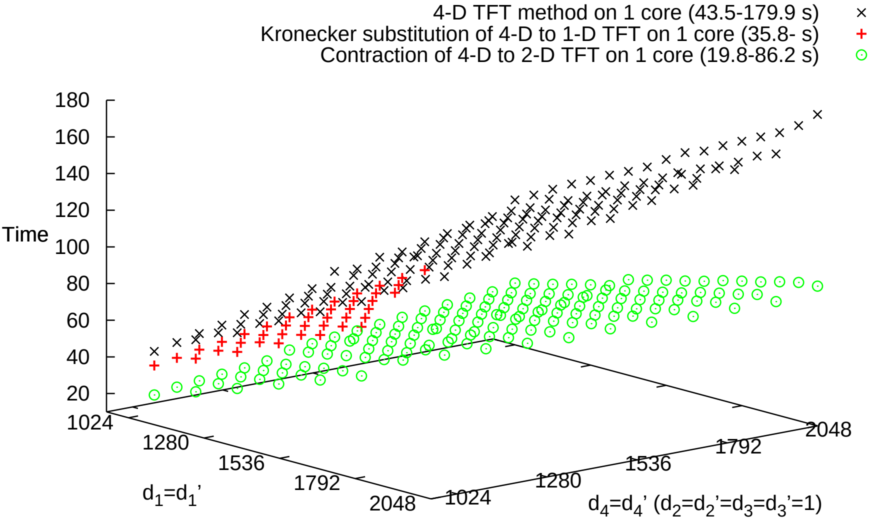
# Performance of Contraction (timing)



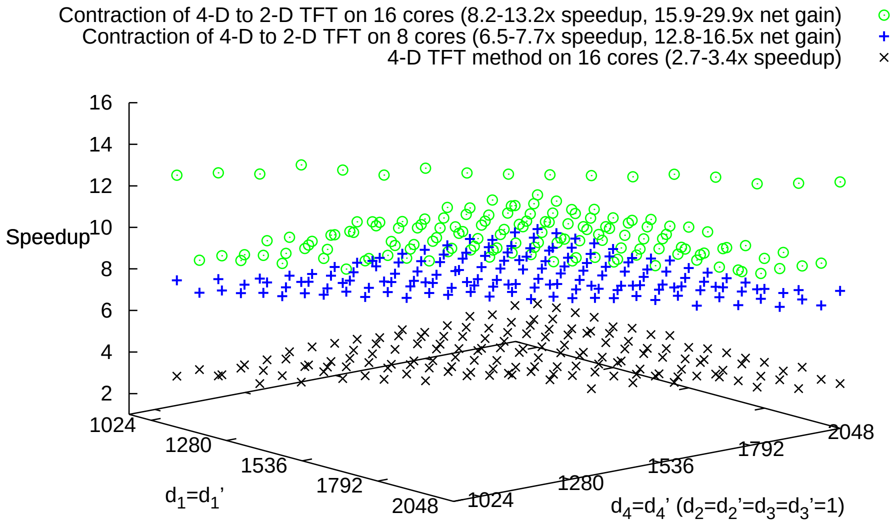
# Performance of Contraction (speedup)



# Performance of Contraction for a Large Range of Problems (timing on 1 processor)



# Performance of Contraction for a Large Range of Problems (speedup)



# 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

$$\boldsymbol{\Phi_b} : \mathbf{x^e} \longmapsto \mathbf{u}^{\mathbf{e} \,\mathrm{rem}\, \mathbf{b}} \; \mathbf{v}^{\mathbf{e} \,\mathrm{quo}\, \mathbf{b}}.$$

# 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.

# 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.

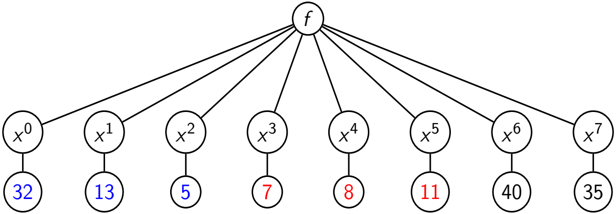
- We compute an integer $b$, such that $fg$ can be performed via $f_b g_b$ using "nearly square" 2-D FFTs, where $f_b := \Phi_b(f)$, $g_b := \Phi_b(g)$ and

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

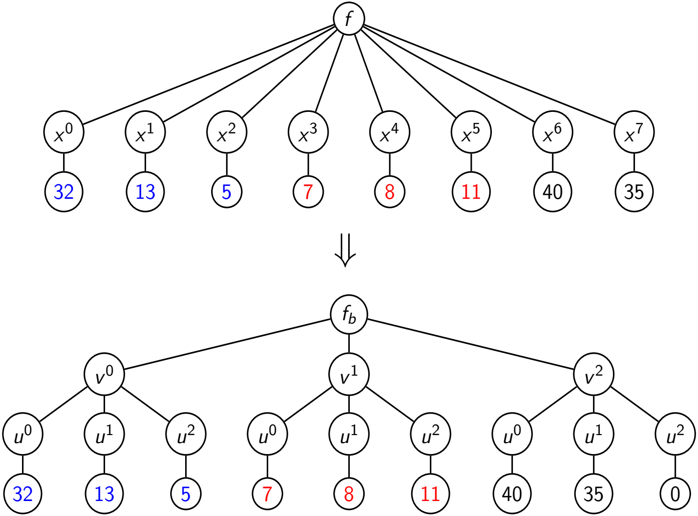
⋆ Here $b = 3$ works since $\deg(f_b g_b, u) = \deg(f_b g_b, v) = 4$; moreover the dense size of $f_b g_b$ is 25.

**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

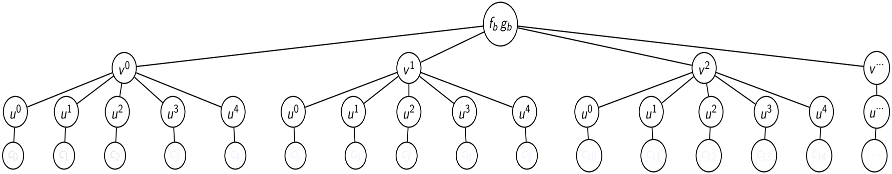


# Extension of $f(x)$ to $f_b(u, v)$ in Recursive Dense Representation



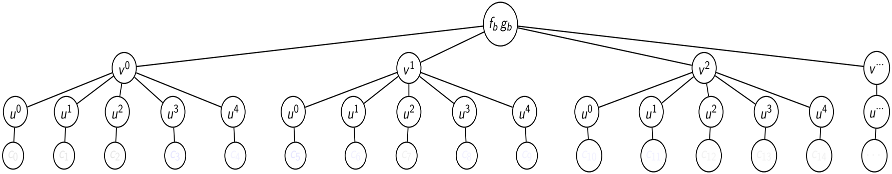
# Conversion to Univariate from the Bivariate Product

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

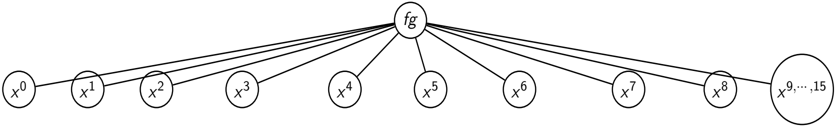


# Conversion to Univariate from the Bivariate Product

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

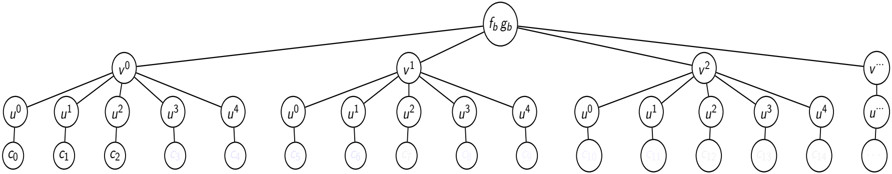



- Convert to univariate: $deg(fg, x) = 15.$

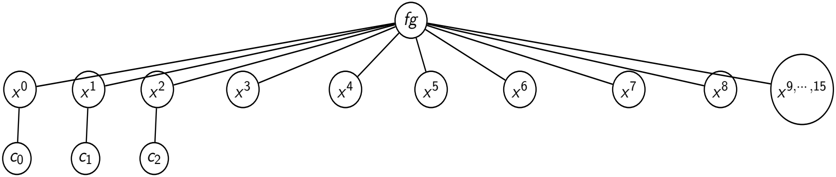


# Conversion to Univariate from the Bivariate Product

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

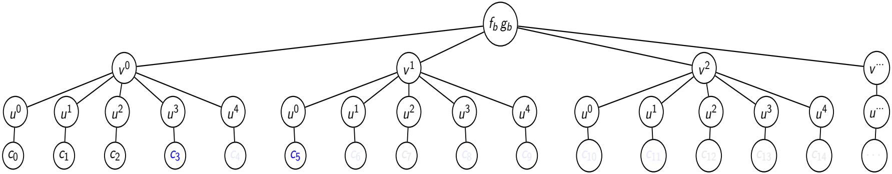



- Convert to univariate: $deg(fg, x) = 15.$

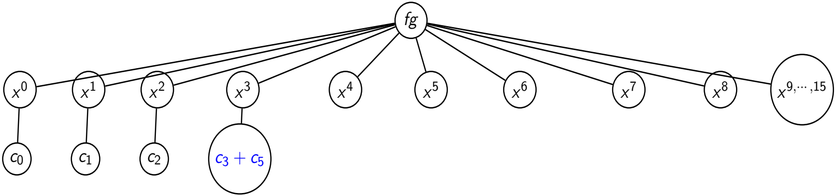


# Conversion to Univariate from the Bivariate Product

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

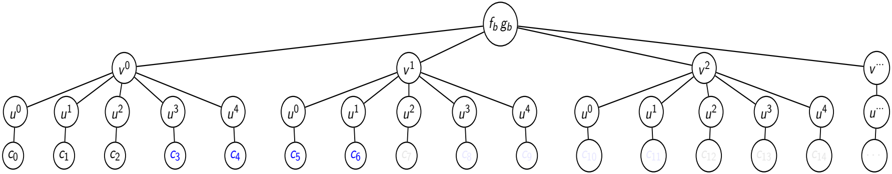



- Convert to univariate: $deg(fg, x) = 15.$

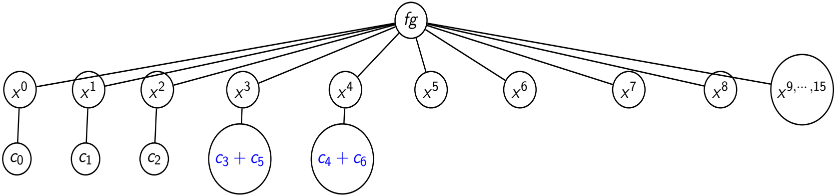


# Conversion to Univariate from the Bivariate Product

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

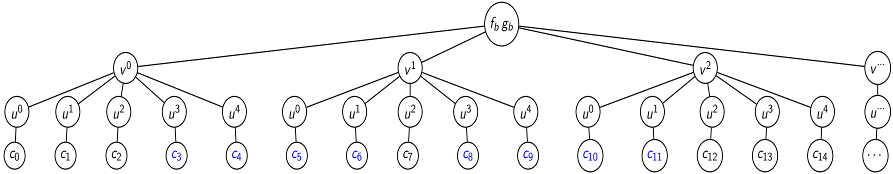



- Convert to univariate: $deg(fg, x) = 15.$

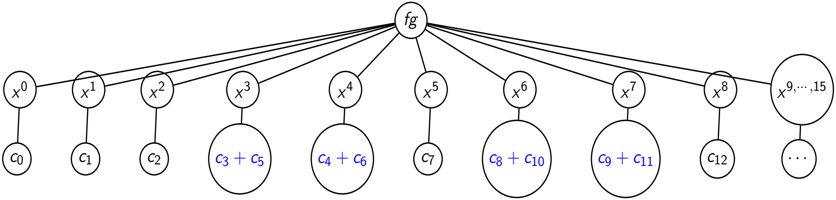


# Conversion to Univariate from the Bivariate Product

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

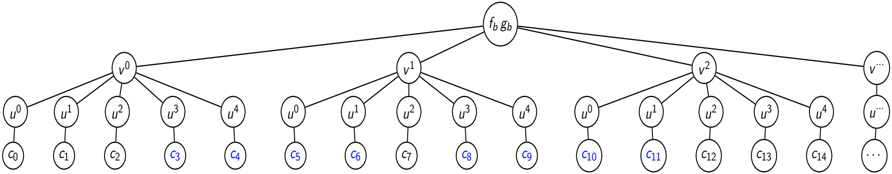



- Convert to univariate: $deg(fg, x) = 15.$

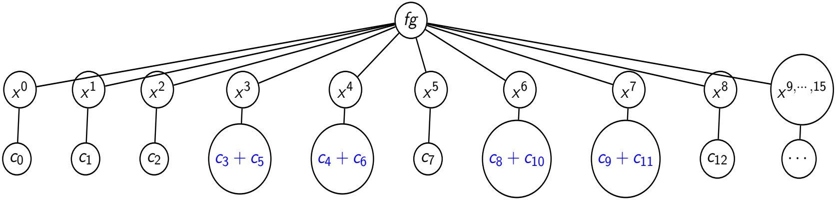


# Conversion to Univariate from the Bivariate Product

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




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

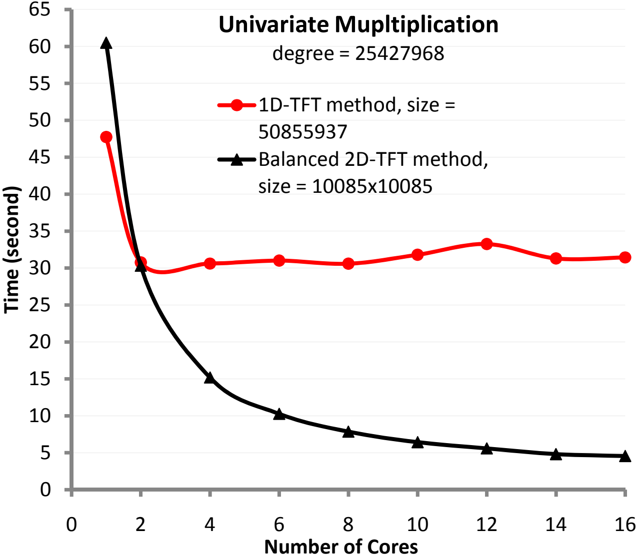



**Remark 4:** Converting back to $fg$ from $f_b g_b$ requires only to traverse the coefficient array once, and perform at most $\deg(fg, x)$ additions.

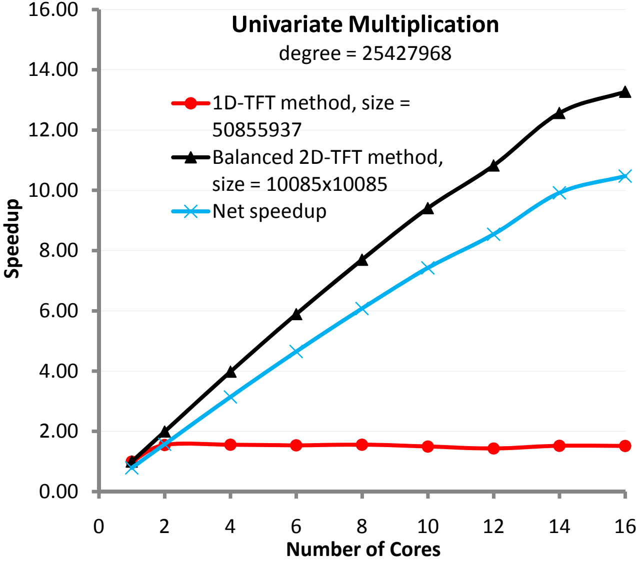
# Remark 3

Our extension technique provides an alternative to the Schönage -Strassen Algorithm for handling problems which sizes are too large for the available primitive roots of unity, a limitation with FFTs over finite fields.

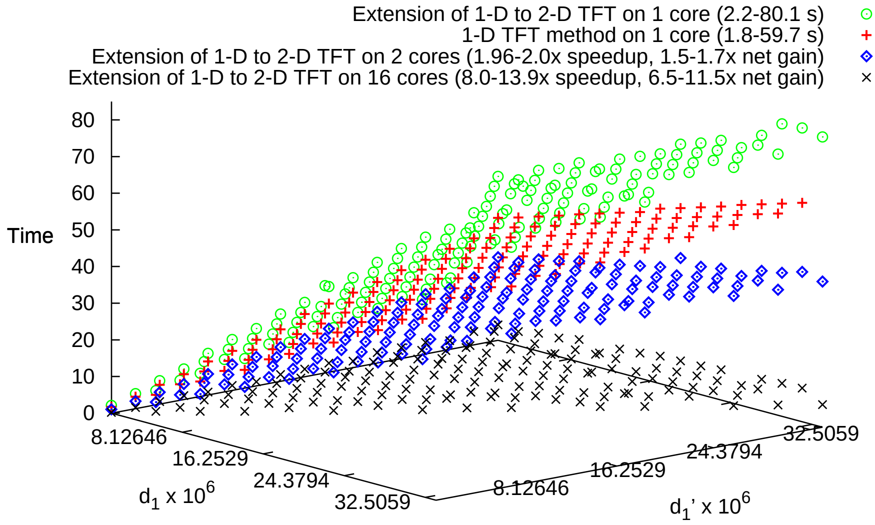
# Performance of Extension (timing)



# Performance of Extension (speedup)



# Performance of Extension for a Large Range of Problems (timing)



# Solution 3: Balanced Multiplication

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

# 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 $v$.

# Solution 3: Balanced Multiplication

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

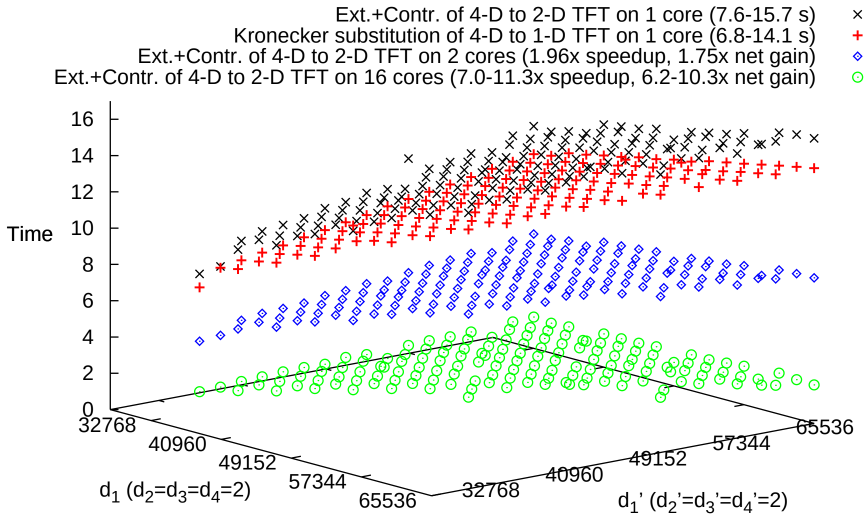
**Algorithm**. Let $f, g \in \mathbf{k}[x_1 < \ldots < x_n]$. W.l.o.g. one can assume $d_1 >> d_i$ and $d_1{}' >> d_i$ for $2 \le i \le n$ (up to variable re-ordering and contraction). Then we obtain $fg$ by

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

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

**Remark 5:** The above extension $\Phi_b$ 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 Multiplication for a Large Range of Problems (timing)



# Conclusion

**Summary and future work:**

- We have obtained efficient techniques and implementations for dense polynomial multiplication on multicores:
  - use balanced bivariate multiplication as a kernel,
  - contract mutivariate to bivariate,
  - extend univariate to bivariate,
  - combine contraction and extension.
- Our work-in-progress include normal form, GCD/resultant 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 at MIT, Dr. Matteo Frigo and all other members of Cilk Arts.

Thanks to all who have been supporting!