Balanced Dense Polynomial Multiplication on Multicores

Marc Moreno Maza† and Yuzhen Xie⋆

†Ontario Research Centre of Computer Algebra, University of Western Ontario, London, Canada
⋆Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139, USA

Overview

In symbolic computation, polynomial multiplication is a fundamental operation akin to matrix multiplication in numerical computation. We present efficient implementation strategies for FFT-based dense polynomial multiplication targeting multi-cores. We show that balanced input data can maximize parallel speed-up and minimize cache complexity for bivariate multiplication. However, unbalanced input data, which are common in symbolic computation, are challenging. We provide efficient techniques that we call *contraction* and *extension* to reduce multivariate (and univariate) multiplication to balanced bivariate multiplication. Our implementation in `Cilk++` demonstrates good speed-up on multi-cores.

FFT-based Multivariate Multiplication

Let $\mathbb{K}$ be a field and $f, g \in \mathbb{K}[x_1 < \cdots < x_n]$ 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 $\omega_i \in \mathbb{K}$, 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 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.

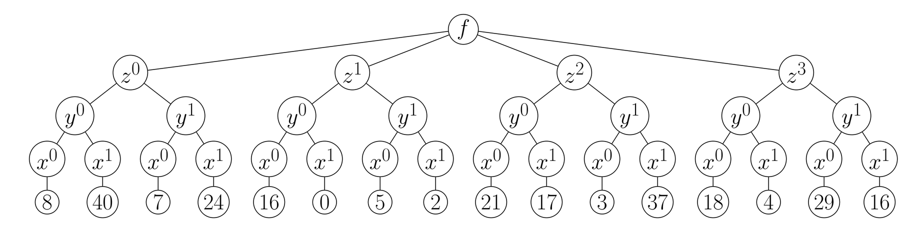
Complexity Estimates

- Let $s = s_1 \cdots s_n$. The number of operations in $\mathbb{K}$ for computing fg based on FFTs 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 serial 1-D FFT assumption, the span of *Step* 1 is $\frac{9}{2}(s_1\lg(s_1) + \cdots + s_n \lg(s_n))$, and the parallelism of *Step* 1 is lower bounded by
$$s/\max(s_1, \ldots, s_n). \qquad (1)$$
- 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
$$n\frac{cs}{L} + cs(\frac{1}{s_1} + \cdots + \frac{1}{s_n}). \qquad (2)$$

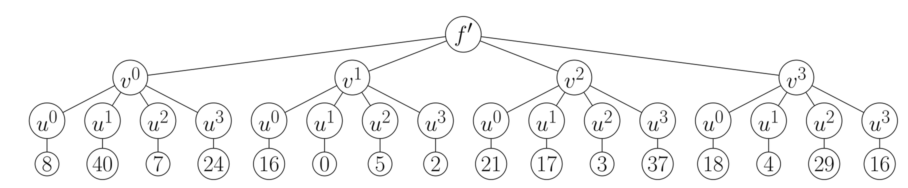
- **Remark**: For $n \geq 2$, Expr. (2) is minimized at $n = 2$ and $s_1 = s_2 = \sqrt{s}$. Moreover, when $n = 2$, under a fixed $s = s_1 s_2$, Expr. (1) is maximized at $s_1 = s_2 = \sqrt{s}$.

Contraction to Bivariate

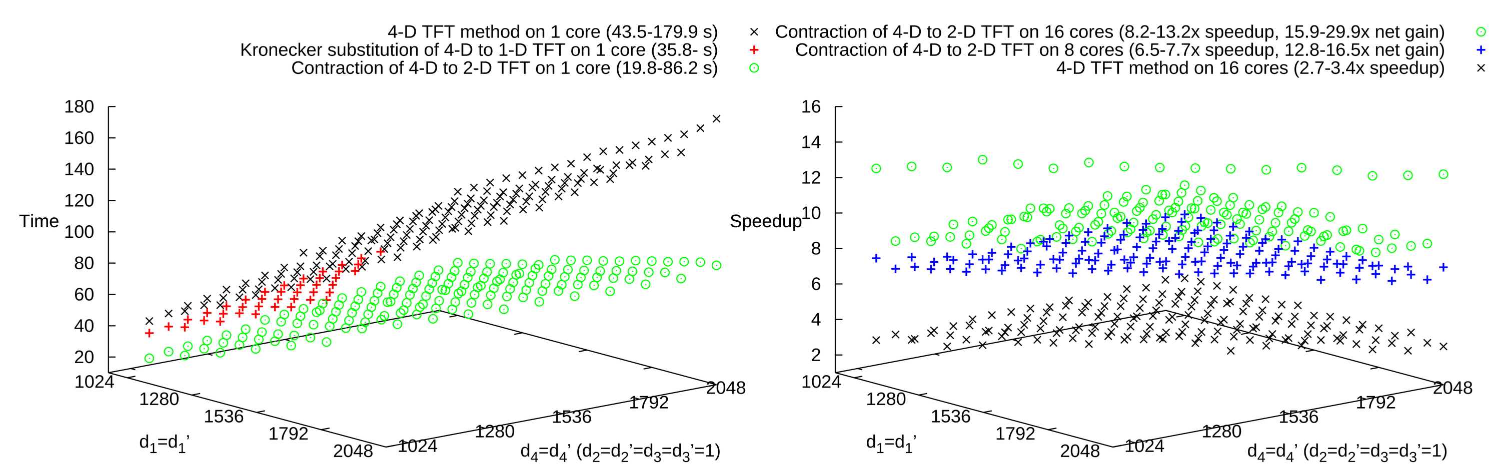
- **Example**. Let $f \in \mathbb{K}[x, y, z]$ where $\mathbb{K} = \mathbb{Z}/41\mathbb{Z}$, with $\deg(f, x) = \deg(f, y) = 1$, $\deg(f, z) = 3$ and recursive dense representation:



Contracting $f(x, y, z)$ to $f'(u, v)$ by $x^{e_1}y^{e_2} \mapsto u^{e_1+2e_2}$, $z^{e_3} \mapsto v^{e_3}$:

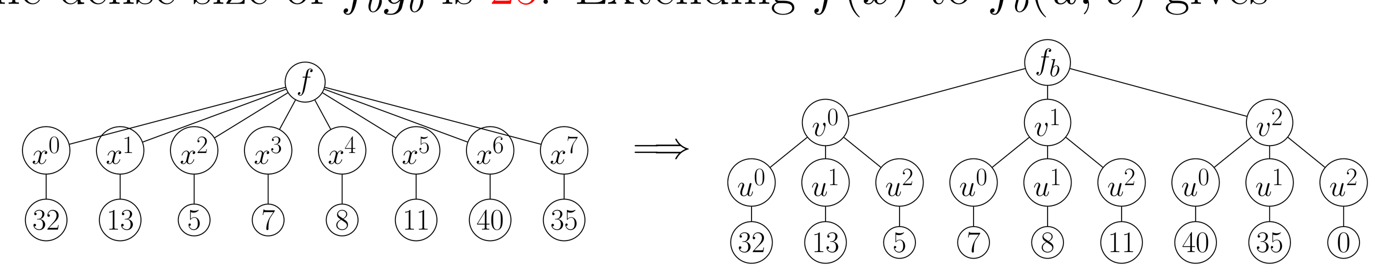


- **Remark**. The data is "essentially" unchanged by contraction, which is a property of recursive dense representation.
- Below, the left figure displays the timing of 4-variate multiplication via 4-D TFT, 1-D TFT by Kronecker substitution and contraction to balanced 2-D TFT on 1 core; The right figure shows the speedups of 4-variate multiplication using 4-D TFT and contraction to balanced 2-D TFT on 8 and 16 cores.

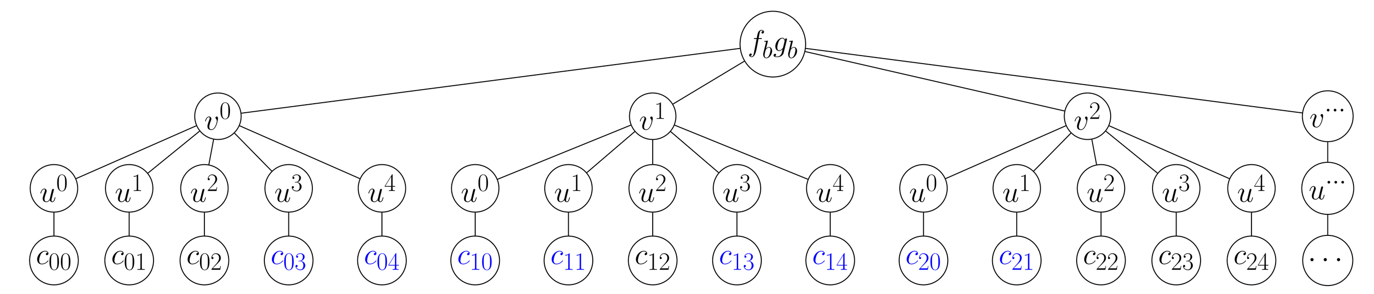


Extension from Univariate to Bivariate

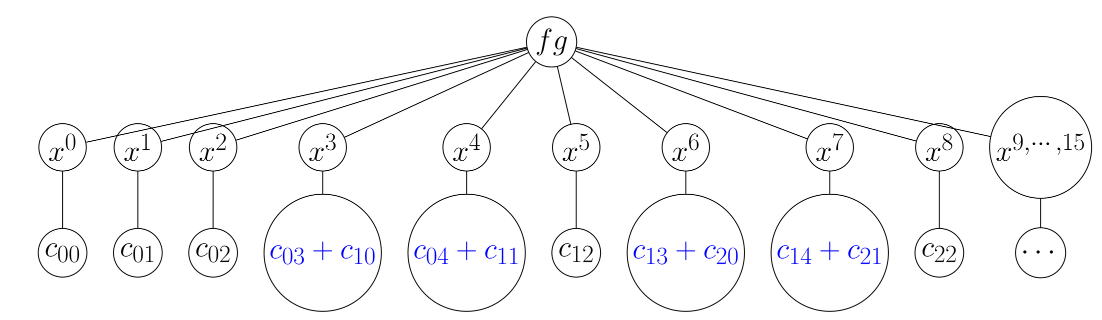
- **Example**: Consider $f, g \in \mathbb{K}[x]$ univariate, with $\deg(f) = 7$ and $\deg(g) = 8$; fg has "dense size" 16. We obtain an integer b, such that fg can be performed via $f_b g_b$ using "nearly square" 2-D FFTs, where $f_b := \Phi_b(f)$, $g_b := \Phi_b(g)$ and
$$\Phi_b : \ x^e \longmapsto u^{e \text{ rem } b}\, v^{e \text{ quo } b}.$$
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. Extending $f(x)$ to $f_b(u, v)$ gives



- **Proposition**: For any non-constant $f, g \in \mathbb{K}[x]$, 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.
- **Example (ctnd)**: Computing the bivariate product $f_b g_b$:



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.



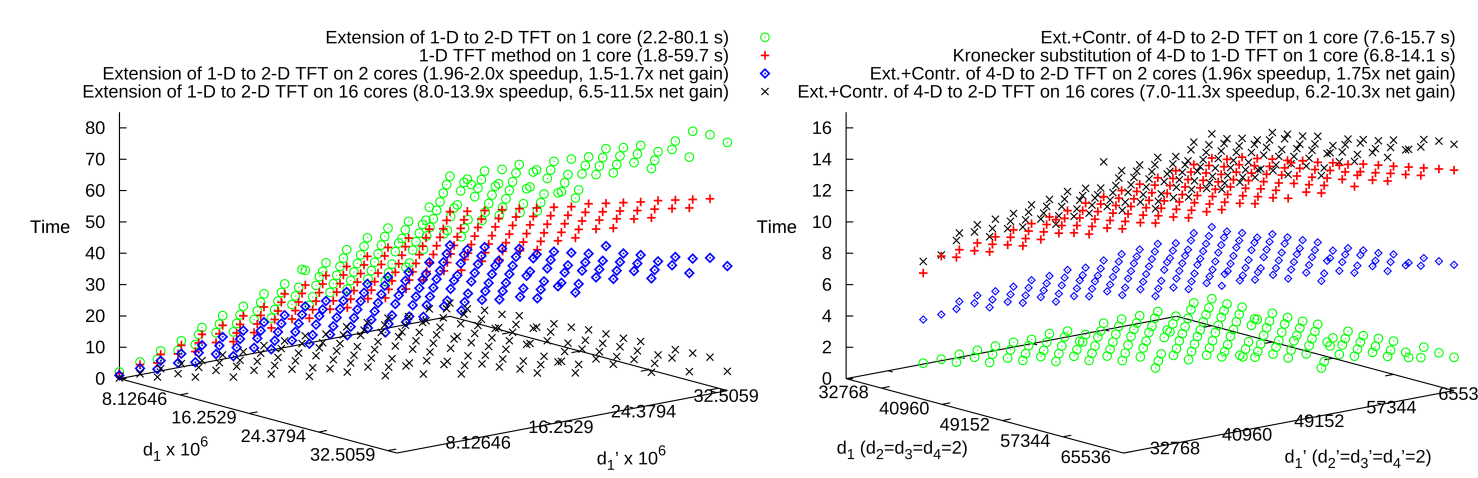
Balanced Multiplication

- **Definition**. A pair of bivariate polynomials $p, q \in \mathbb{K}[u, v]$ is *balanced* if $\deg(p, u) + \deg(q, u) = \deg(p, v) + \deg(q, v)$.
- **Algorithm**. Let $f, g \in \mathbb{K}[x_1 < \ldots < x_n]$. 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). We obtain fg by

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

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

  Determine the above extension Φ_b 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.
- The left figure shows the timing of univariate multiplication via 1-D TFT and extension to balanced 2-D TFT on 1, 2, 16 cores; The right one shows the timing of our balanced multiplication for an unbalanced 4-variate case on 1, 2, 16 cores vs the method based on 1-D TFT via Kronecker substitution.



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 CSAIL MIT and Cilk Arts.