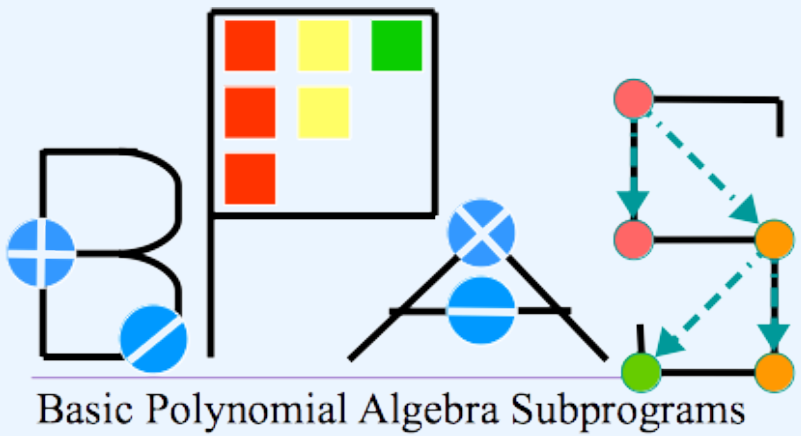


Parallel Integer Polynomial Multiplication

Changbo Chen[1] Svyatoslav Covanov[2,3] Farnam Mansouri[2]
Marc Moreno Maza[2] Ning Xie[2] Yuzhen Xie[2]

[1]Chinese Academy of Sciences, China
[2]University of Western Ontario, Canada
[3]LORIA, Universté de Lorraine, France

SYNASC, West University of Timisoara, September 24, 2016

Overview

- Polynomial multiplication is at the core of many algorithms in symbolic computation.
- Classical (but asymptotically fast) algorithms for multiplying dense integer polynomials (Toom-Cook, Schönaghe-Strassen) are hard to parallelize on multi-core architectures
- It is, therefore, natural to consider reducing computations from $\mathbb{Z}[x]$ to $\mathbb{F}_p[x, y]$, which allows to use 2D FFTs.
- For a well-chosen prime number p, this strategy leads not only to a practically efficient parallel algorithm but also to a nice complexity estimate for the work (i.e. arithmetic count).

Plan

1. Dense polynomial multiplication: classical algorithms
2. The two-convolution method
3. Experimentation

Schönaghe-Strassen via Kronecker's substitution

0 **Input**: $f = \sum_{i=0}^{n} f_i\, x^i$ and $g = \sum_{i=0}^{m} g_i\, x^i$

1 **Choose**: $2^\ell \geq \|f\|_\infty + \|g\|_\infty + \max(n, m) + 1$

2 **Evaluation**: $Z_f = \sum_{i=0}^{n} f_i\, 2^{i\ell}$ and $Z_g = \sum_{i=0}^{m} g_i\, 2^{i\ell}$;

3 **Multiplying**: $Z_h = Z_f \times Z_g$, using GMP library;

4 **Unpacking**: h_i from $Z_h = \sum_{i=0}^{n+m} h_i\, 2^{i\ell}$.

5 **Return**: $f\, g = \sum_{i=0}^{n+m} h_i\, x^i$

- its work in terms of bit operations is $O(s \log_2(s) \log_2(\log_2(s)))$, where s is the maximum bit-size of f or g;
- purely serial due to the difficulties of parallelizing 1-D FFTs on multicore processors.

D-n-C with reduction to GMP's integer multiplication

1 **Division**: $f(x) = f_0(x) + f_1(x)\,x^{n/2}$ and $g(x) = g_0(x) + g_1(x)\,x^{n/2}$;

2 **Execute recursively**:

Store $f_0 \times g_0$ & $f_1 \times g_1$ in the result array;
Store $f_0 \times g_1$ & $f_1 \times g_0$ in the auxiliary arrays;

3 **Addition**: add the auxiliary arrays to the result one.

- use (one or) two levels of recursion, then use the KS+SS algorithm;
- its work in terms of bit operations is $O(s \log_2(s) \log_2(\log_2(s)))$, where s is the maximum bit-size of f or g, but the constant has been multiplied approximately by 4;
- static parallelism (close to 16).

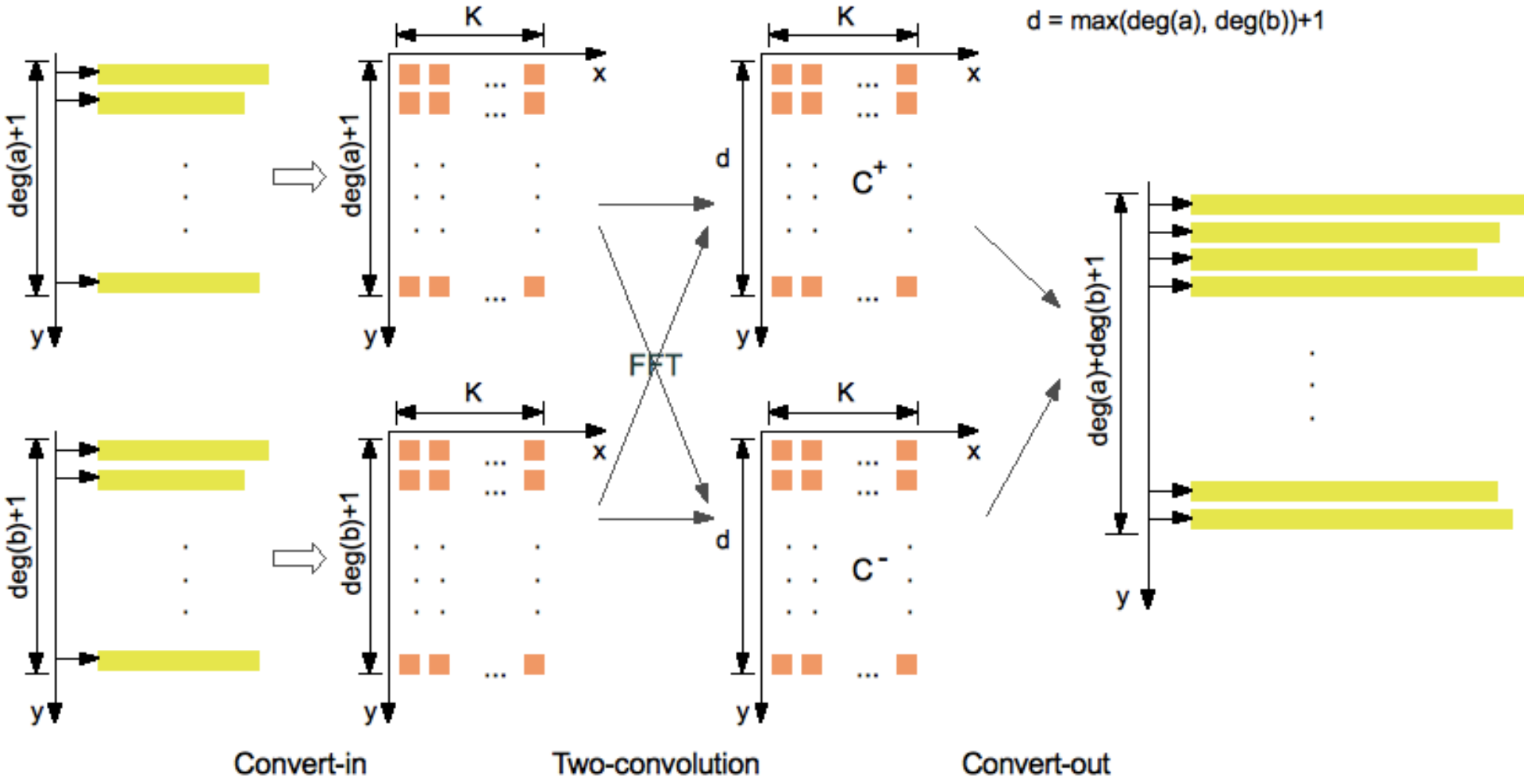
k-way Toom-Cook algorithms

1. **Division**: $f(x) = f_0(x) + f_1(x)\,x^{n/k} + \cdots + f_{k-1}(x)\,x^{(k-1)n/k}$ and $g(x) = g_0(x) + g_1(x)\,x^{n/k} + \cdots + g_{k-1}(x)\,x^{(k-1)n/k}$;
2. **Conversion**: Set $X = x^{n/k}$ and obtain $F(X) = Z_{f_0} + Z_{f_1}\,X + \cdots + Z_{f_{k-1}}\,X^{k-1}$ and $G(X) = Z_{g_0} + Z_{g_1}\,X + \cdots + Z_{g_{k-1}}\,X^{k-1}$;
3. **Evaluation**: Evaluate f, g at $2\,k-1$ points: $(0, X_1, \ldots, X_{2\,k-3}, \infty)$;
4. **Multiplying**: $(w_0, \ldots, w_{2\,k-2}) = (F(0)\cdot G(0), \ldots, F(\infty)\cdot G(\infty))$;
5. **Interpolation**: Recover $(Z_{h_0}, Z_{h_1}, \ldots, Z_{h_{2\,k-2}})$ where $H(X) = f(X)\,g(X) = Z_{h_0} + Z_{h_1}\,X + \cdots + Z_{h_{2\,k-2}}\,X^{2\,k-2}$
6. **Conversion**: Recover polynomial coefficients from $Z_{h_0}, \ldots, Z_{h_{2\,k-2}}$, obtaining $h(x) = h_0(x) + h_1(x)\,x^{n/k} + \cdots + h_{2\,k-2}(x)\,x^{(2\,k-2)\,n/k}$.

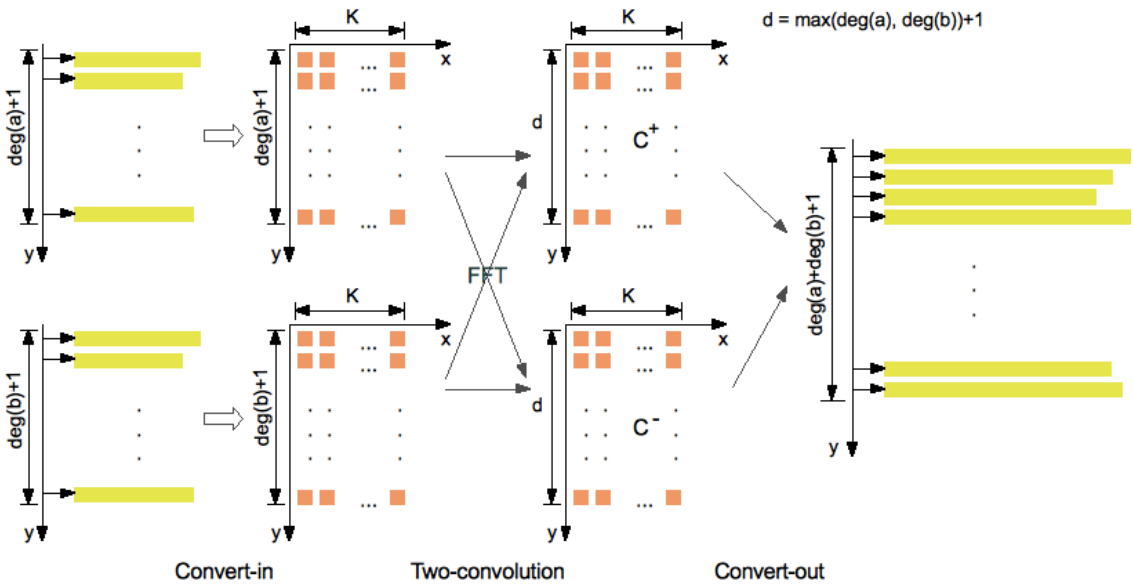
- work in terms of bit operations is $O(s\,\log_2(s)\,\log_2(\log_2(s)))$, where s is the maximum bit-size of f or g, but the constant has been multiplied approximately by 2 for k $= 8$;
- 4-way & 8-way Toom-Cook are available;
- static parallelism (about 7 and 13 when k $= 4$ and k $= 8$, resp).

Plan

From $\mathbb{Z}[x]$ to $\mathbb{F}_p[x, y]$ allowing 2D FFT



Cyclic and nega-cyclic convolutions to recover coefficients



1. Convert $a(y)$, $b(y)$ to bivariate $A(x,y)$, $B(x,y)$ s. t. $a(y) = A(\beta, y)$ and $b(y) = B(\beta, y)$ hold at $\beta = 2^M$, $K = \deg(A,x) = \deg(B,x)$, where KM is essentially the maximum bit size of a coefficient in a and b.
2. Consider $C^+(x,y) \equiv A(x,y)\,B(x,y) \bmod < x^K + 1 >$ and $C^-(x,y) \equiv A(x,y)\,B(x,y) \bmod < x^K - 1 >$, then compute $C^+(x,y)$ and $C^-(x,y)$ modulo a (sufficiently large) prime so as to use efficient 2-D FFTs.
3. Consider $C(x,y) = \frac{C^+(x,y)}{2}(x^K - 1) + \frac{C^-(x,y)}{2}(x^K + 1)$, then evaluate $C(x,y)$ at $x = \beta$, which finally yields the product $c(y) := a(y)\,b(y)$.

Complexity estimates (1/3)

Notations

- $a(y), b(y) \in \mathbb{Z}[y]$ with $d = \max(\deg(a), \deg(b)) + 1$.
- $N \in \mathbb{N}$ s.t. each coefficient of $a(y), b(y)$ writes within N bits
- $K, M \in \mathbb{N}$ s.t. $N = KM$.

Assumptions

- $K \in \Theta(d)$ and,
- $M \in \Theta(\log d)$.

Results

The two-convolution method multiplies $a(y)$ and $b(y)$ with

- a work of $O(dKM\log(dK)\log\log(\log(d)))$ word operations,
- a span of $O(\log_2(d)KM)$ word operations and,
- incurs $O(1 + (dMK/L)(1 + \log_Z(dMK)))$ cache misses.

Complexity estimates (2/3)

Recall

Assuming:

- $a(y), b(y) \in \mathbb{Z}[y]$ with $d = \max(\deg(a), \deg(b)) + 1$,
- each coefficient of $a(y), b(y)$ writes within $N = K\,M$ bits,
- $K \in \Theta(d)$ and $M \in \Theta(\log d)$.

Then, $a(y)$ and $b(y)$ can be multiplied

- within $O(dKM\log(dK)\log\log(\log(d)))$ word operations,
- thus within $O(d\,N\,\log(d^2)\,\log\log(\log(d)))$

Comments

- The assumptions $K \in \Theta(d)$ and $M \in \Theta(\log d)$ can be met by *balancing techniques* (generalization of Kronecker's substitution) see (M. Moreno Maza & Y. Xie, Int. J. Found. Comput. Sci., 2011)
- Our result is better than that of Schönhage & Strassen which gives here $O(d\,N\,\log(dN)\,\log(\log(dN)))$.

Complexity estimates (3/3)

Key argument in the analysis

Let w be the bit-size of a machine word. Then, one can choose p (and thus $e := \lfloor \log_w(p) \rfloor + 1$) such that computing an FFT of a vector of size s over $\mathbb{F}_p[x]$, amounts to

$$\mathsf{F}_{\mathrm{word}}(e, s) \in O(s\, e \log(s) \log\log(e)) \tag{1}$$

machine-word operations, whenever $e \in \Theta(\log s)$ holds and p is a *generalized Fermat prime*, e.g. $(2^{63} + 2^{34})^8 + 1$.

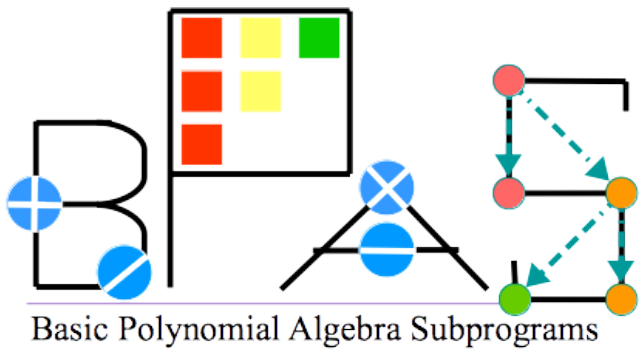
In practice ...

- We have $e \geq \left\lceil \frac{2+\lceil \log_2(d\,K) \rceil + 2M}{w} \right\rceil$.
- Efficiently implementing arithmetic operations in $\mathbb{F}_p[x]$ for a generalized Fermat prime p is *work in progress* on multi-core architectures (while already successful on GPUs).
- Using Fourier primes instead of generalized Fermat primes (on multi-core architectures) makes the work of the two-convolution slightly higher than that of Schönhage & Strassen, which is verified experimentally.

Plan

Implementation

- The two-convolution methods (as well as KS + SS, D-n-C, Toom_4, Toom_8) are implemented in `CilkPlus` targeting multi-core architectures.
- Moreover, those algorithms are combined in an adaptive algorithm.
- We compare this latter against `FLINT` and MAPLE.
- From $e \geq \left\lceil \frac{2+\lceil \log_2(d\,K)\rceil+2M}{w} \right\rceil$, it follows that on today's computers (say for input data size in order of giga-bytes) it is sufficient to have $1 \leq e \leq 8$.
- In our implementation, we use machine-word size Fourier primes together with the CRA instead of using a single generalized Fermat prime: fixing this limitation is work in progress.



www.bpaslib.org

Large degrees and coefficients

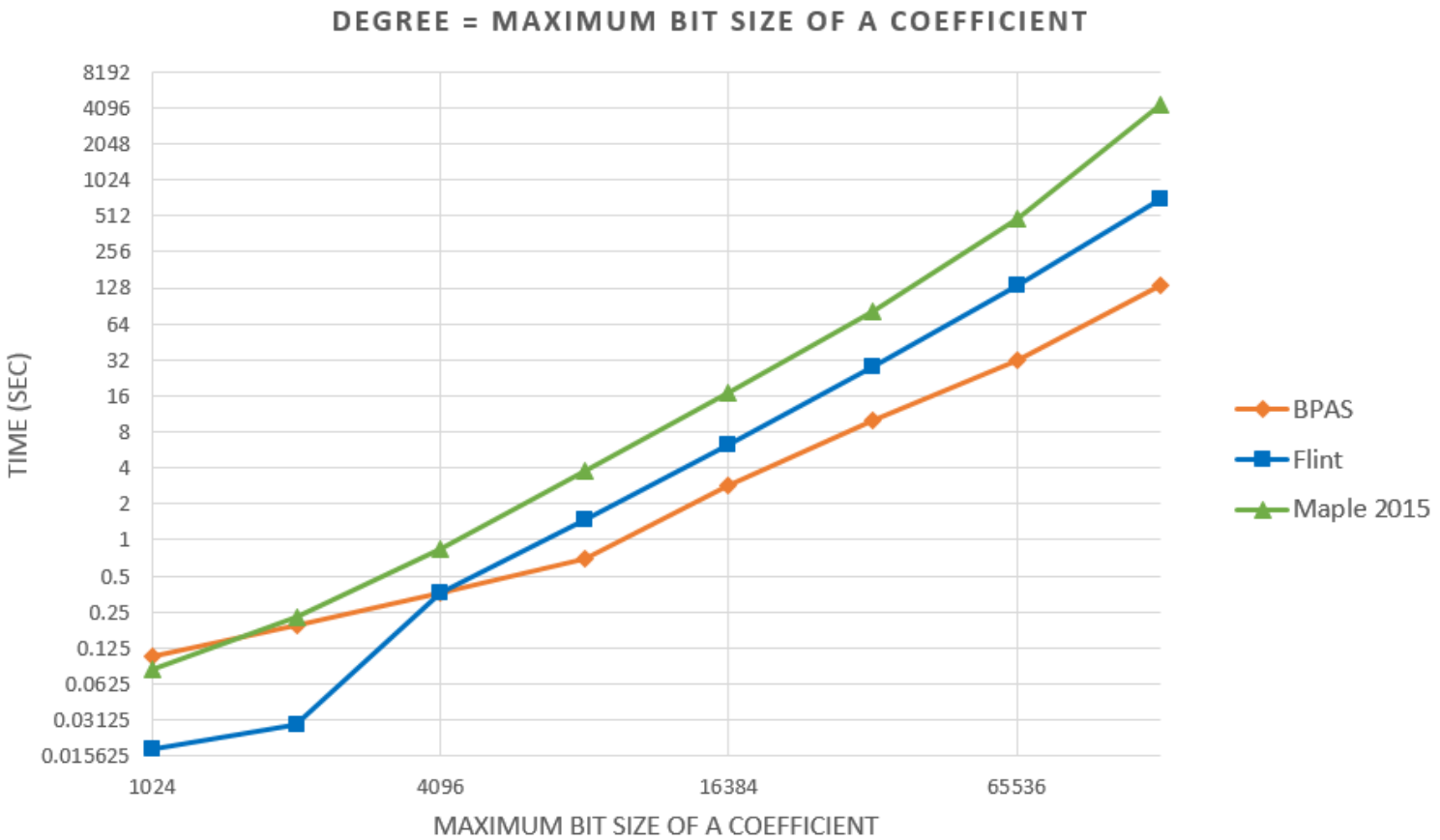


Figure: BPAS (parallel) vs FLINT (serial) vs Maple 2015 (serial) with logarithmic scale in radix 2 of the maximum bit-size of an input polynomial as the horizontal axis

Large coefficients only

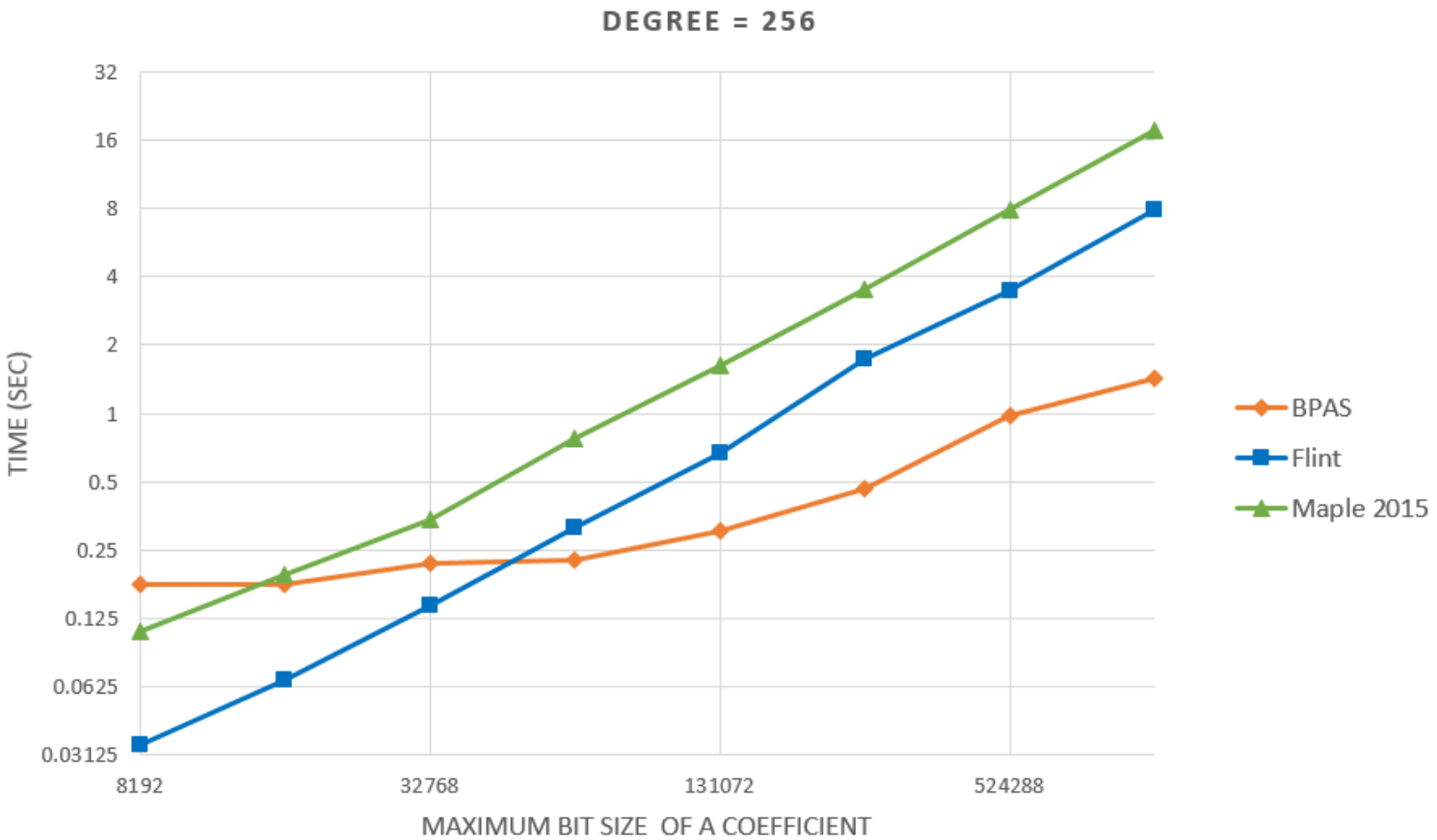


Figure: BPAS (parallel) vs FLINT (serial) vs Maple 2015 (serial) with logarithmic scale in radix 2 of the maximum bit-size of an input polynomial as the horizontal axis

Timings for polynomial multiplication with $d = N$.

d, N	CVL_p^2	DnC_p	Toom_p^4	Toom_p^8	KS_s	FLINT_s	Maple_s^{18}
2^9	0.152	0.049	0.022	0.026	0.018	0.005	0.054
2^{10}	0.139	0.11	0.046	0.059	0.057	0.016	0.06
2^{11}	0.196	0.17	0.17	0.17	0.25	0.067	0.201
2^{12}	0.295	0.58	0.67	0.64	1.37	0.42	0.86
2^{13}	0.699	2.20	2.79	2.73	5.40	1.671	3.775
2^{14}	1.927	8.26	10.29	8.74	20.95	7.178	17.496
2^{15}	9.138	30.75	35.79	33.40	92.03	32.112	84.913
2^{16}	33.04	122.1	129.4	115.9	*Err.	154.69	445.67

The two-convolution scales well

The adaptive algorithm based on the input size and available resources

- Very small: Plain multiplication
- Small or Single-core: Kronecker substitution + Schönhage & Strassen
- Big but a few cores: 4-way Toom-Cook
- Big: 8-way Toom-Cook
- Very big: Two-convolution method

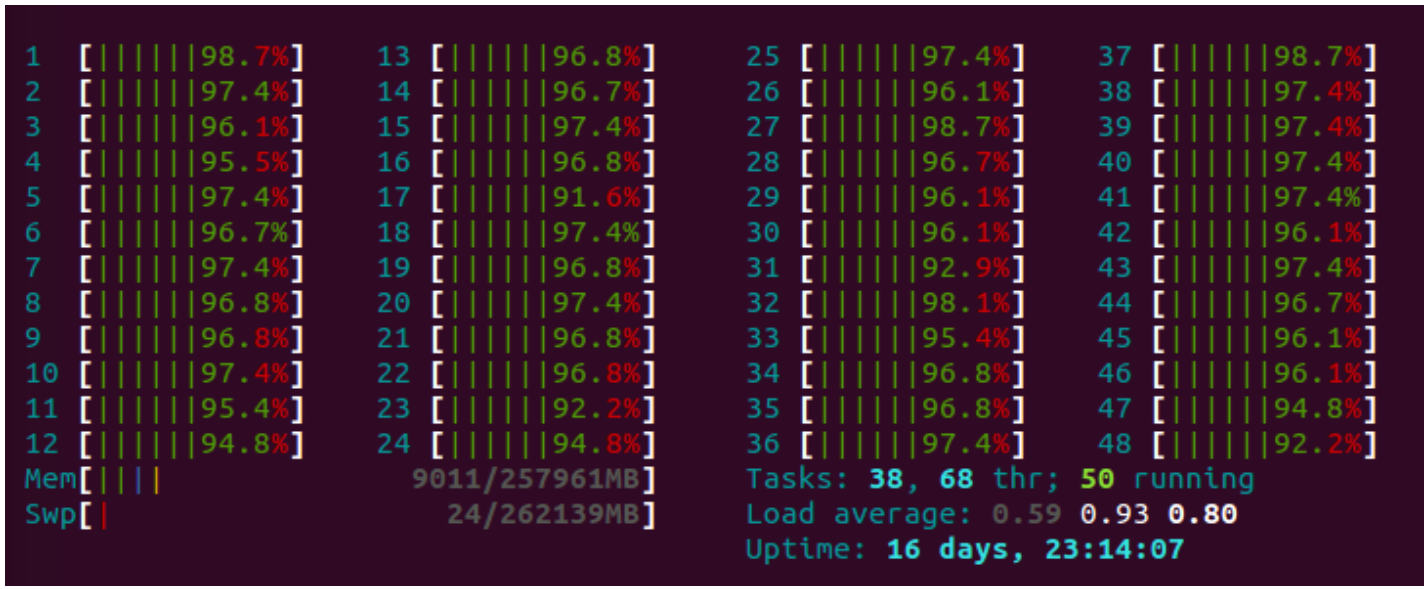
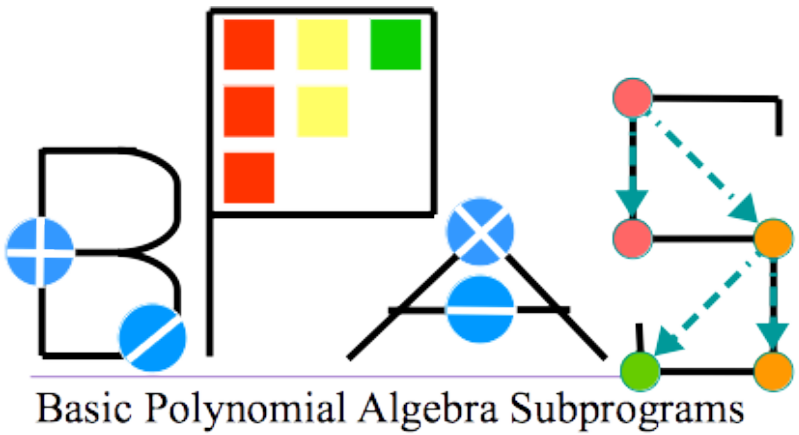


Figure: The `htop` screenshot of multiplying two large integer polynomials in BPAS

Conclusions

- We have proposed a new algorithm for multiplying dense integer polynomials
- Via a transformation from $\mathbb{Z}[x]$ to $\mathbb{F}_p[x, y]$, this algorithm essentially relies on 2D FFTs, which parallelize nicely on multi-core architectures.
- Using for p a generalized Fermat prime (together with ideas borrowed from Fürer's algorithm), this new algorithm outperforms that of Schönhage & Strassen in terms of algebraic complexity
- Our multi-core experimentation shows promising results, though simply using Fourier primes for the moment.



www.bpaslib.org