Complexity and Performance Results for Non FFT-Based Univariate Polynomial Multiplication

M.F.I. Chowdhury, M. Moreno Maza, W. Pan, É. Schost
Ontario Research Centre of Computer Algebra, University of Western Ontario, London, Canada

Overview

Many experimentations reported in the literature show that, for dense univariate polynomials, FFT-based multiplication provides better running times than other techniques such as the plain algorithm and the ones based on the tricks of Toom-Cook and Karatsuba. This remains true on the latest hardware architectures. However, implementation techniques and thus thresholds are certainly changing. In addition, if *coefficient-oblivious* algorithms cannot outperform FFT-based methods for sufficiently large input, these former algorithms are by nature generic and easier to implement. Therefore, it is important to understand how to implement them efficiently on today's parallel hardware architectures.

In this poster, we focus on two algorithms which are independent of the coefficient ring: the *plain* and the Toom-Cook univariate multiplications (for the latter, 6 is assumned to be a unit in the base ring). We analyze their cache complexity and report on experimentation with parallel implementations in `Cilk++`.

Plain univariate multiplication

```
void multPoly1(int *A, int n, int *B, int m, int *C) {
    for (i=0; i<n+m-1; i++) C[i] = 0;
    for (i=0; i<n; i++)
        for (j=0; j<m; j++)  C[i+j] += A[i] * B[j];
}
void multPoly2(int *A, int i, int j,
               int *B, int k, int l, int *C) {
    int da = j-i; int db = l-k;
    if (da == 0) {
        for (int x=k; x<=l; ++x) C[i+x] += A[i] * B[x];
    } else if (db == 0) {
        for (int y=i; y<=j; ++y) C[y+k] += A[y] * B[k];
    } else if (da >= db) {
        int m = (da+1)/2;
        multPoly2(A, i, i + m - 1, B, k, l, C);
        multPoly2(A, i + m, j, B, k, l, C);
    } else {
        int m = (da + 1) / 2;
        multPoly2(A, i, j, B, k, k + m - 1, C);
        multPoly2(A, i, j, B, k + m, l, C);
    } }
```

The above `multPoly1` is a naive pseudo-implementation of the plain multiplication, where the input coefficient arrays `A` and `B` have size `n` and `m`. For an *ideal cache* with Z words amd cache line size L, this algorithm incurs $O(\frac{2mn+m+n-1}{L} + n(2+\frac{1}{L}))$ *cache misses*. The *divide-and-conquer* `multPoly2` reduces this cache complexity to $\Theta(nm/LZ)$.

In practice, even with a threshold between its iterative and recursive modes in the order of 512, `multPoly2` brings a speedup factor limited to 2, due to the overhead of the recursive calls. The same algorithm can be parallelized in a natural manner with a span of $\Theta(n)$ (assuming $n = m$ for simplicity) and a space complexity of $\Theta(n \log n)$.

Toom-Cook univariate multiplications

Our initial goal was to obtain a parallel version of Toom-Cook multiplication for which both span and space would be in $\Theta(n)$. The procedure below sketches the principle of a solution for input polynomials A and B of degree less than n. The other arguments are three arrays R, *tmpr1* and *tmpr2* each of size $2n$: the first one is meant for storing the product of A and B whereas *tmpr1* and *tmpr2* are auxiliary buffers. Therefore, this procedure runs in space $9n$. This result is to the price of slightly increasing the work. Indeed one needs to performs more *linear combinations* of the segments $A_0, A_1, A_2, B_0, B_1, B_2$ than with a straightforward parallelization of Toom-Cook Algorithm. However, the latter incurs a space consumption within $\Theta(n^{\log_5 3})$ (in bytes), that is, in the order of the work.

Algorithm 1 ParaTCMul$(A, B, n, R, tmpr1, tmpr2)$

Require: 2 polynomials $A = A_0(x)+A_1(x)x^{\frac{n}{3}}+A_2(x)x^{\frac{2n}{3}}$ and $B = B_0(x)+B_1(x)x^{\frac{n}{2}}+B_2(x)x^{\frac{n}{3}}$
Ensure: The product AB

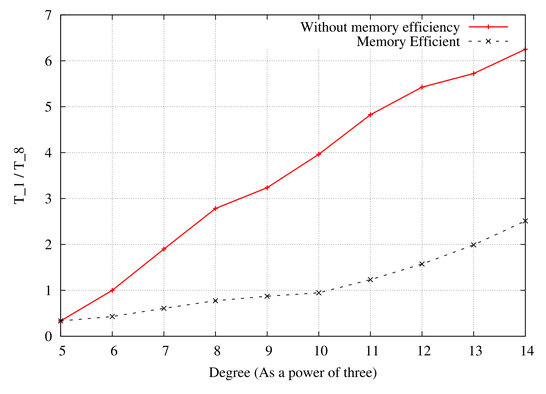
1: **if** $n = 3$ **then**
2: **Return** AB
3: **else**
4: $A_1 = A_0 + A_1 + A_2$; $B_1 = B_0 + B_1 + B_2$;
5: ParaTCMul$(A_0, B_0, \frac{n}{3}, R, tmpr1, tmpr2)$
6: Spawn ParaTCMul$(A_1, B_1, \frac{n}{3}, R + \frac{2n}{3}, tmpr1 + \frac{2n}{3}, tmpr2 + \frac{2n}{3})$
7: Spawn ParaTCMul$(A_2, B_2, \frac{n}{3}, R + \frac{4n}{3}, tmpr1 + \frac{4n}{3}, tmpr2 + \frac{4n}{3})$
8: Sync;
9: $A_1 = 2A_0 + 2A_2 - A_1$; $B_1 = 2B_0 + 2B_2 - B_1$;
10: $A_2 = -A_0 + 2A_1 + 2A_2$; $B_2 = -B_0 + 2B_1 + 2B_2$
11: ParaTCMul$(A_1, B_1, \frac{n}{3}, tmpr1, tmpr1 + \frac{2n}{3}, tmpr1 + \frac{4n}{3})$
12: Spawn ParaTCMul$(A_2, B_2, \frac{n}{3}, tmpr2, tmpr2 + \frac{2n}{3}, tmpr2 + \frac{4n}{3})$; Sync;
13: Linear algebra to recover the original A_0, A_1, A_2 and B_0, B_1, B_2
14: **Return** Linear combination of R, tmp_{r_1}, tmp_{r_2}
15: **end if**

Cache complexity and experimentation

The table below reports timings in milliseconds for multiplying two dense univariate polynomials of degree less than $n = 2^k$. We use a desktop machine (Intel Core 2 Quad @ 2.66GHz) with a L2 cache of standard size (3 Mb). The base size is 512 for both serial codes multPoly2 and multPoly2_p. The speedup of the parallel version multPoly2_p is around 3 comparing to multPoly2.

$\log_2(n)$	multPloy1	multPloy2	multPloy2_p
12	38	19	10
14	535	311	104
16	8570	5049	1679
18	145916	83408	26709
20	3333589	1276392	424464

We have benchmarked the above version Toom Cook Algorithm against a straightforward parallel implementation (span $\Theta(n)$ and space $\Theta(n^{\log_3 3})$) on a 16 cores (each core is an Intel Xeon @ 2.40GHz with 4096 KB of cache). In the figure below, the degree of the input polynomials is given on a logarithmic scale while the vertical coordinate is the speedup factor. Finally, we have proved that the cache complexity of both implementations is within $O\Big(\big(\frac{n}{Z}\big)^{\log_5 3}\big(1+\frac{Z}{L}\big)\Big)$.



In conclusion, coefficient-oblivious univariate multiplication algorithms offer opportunities for improvement in terms of cache and space complexity. However, turning those improvements into practical benefits is challenging. In particular, space saving implementations may reduce parallelism in a dramatic manner.

Acknowledgements. We are very grateful for the help of Dr. Yuzhen Xie and S. A. Haque at ORCCA lab of University of Western Ontario.