Parallel Algorithms for Polynomials

NSERC Discovery Grant Research Proposal, October 2013

My research area is known as Computer Algebra and Symbolic Computation.
I propose to work on the following problems in this area:

1. Polynomial factorization over algebraic number and function fields.
2. Parallel algorithms for polynomial $\times$ and $\div$ modulo triangular sets.
3. Multivariate polynomial interpolation.
4. Parallel algorithms for polynomial GCD and matrices with parameters.

A primary focus will be the development of parallel algorithms and parallel libraries. I am using Cilk. Five years ago when I proposed to shift my research focus to this area, multi-core desktops and servers were becoming the norm for users and research in Computer Algebra needed to address that reality. At that time Computer Algebra Systems like Magma, Maple, Mathematica and Singular were all large serial systems. I thought it would be very difficult to retrofit them to make them parallel systems. Should we instead design a new system that is designed for parallel algorithms at the outset? Perhaps someone should. But I did not know enough about parallel algorithm design and implementation to do that. So I proposed then to build a C library of parallel routines for sparse multivariate polynomial arithmetic (we did multivariate multiplication and division over $\mathbb{Z}$ and $\mathbb{Z}_p$) and link these to Maple in much the same spirit that the LinBox library (see Dumas et. al. [7, 6]) is being developed for exact computations in linear algebra.

I thought, that by introducing fast multiplication and division into Maple, we would get a speedup across the system, in particular, that polynomial factorization would get a boost. My work with Roman Pearce in [21, 22, 23, 24] shows that this is indeed the case. We have achieved a tremendous improvement. Our parallel software is now the fastest available for sparse polynomial multiplication and division and Maple's multivariate factorization code now runs faster than any other system (see [21]).

Currently, my PhD student J. Hu and I are designing a parallel algorithm for polynomial GCD computation in $\mathbb{Z}[x_1, x_2, ..., x_n]$. [The algorithm is linear in the number of terms in the polynomial.] This should benefit any library routines that compute with rational functions, e.g. solving linear systems involving parameters.

I propose to continue the work I started in [13, 14] with my former student M. Javadi on algorithms for polynomial factorization over algebraic number and function fields. I propose to develop a parallel library for polynomial arithmetic modulo triangular sets. This is so we can parallelize polynomial multiplication, division and GCD over algebraic number and functions fields. This will improve the performance of polynomial factorization. I believe that the gain will be even greater than the gain we obtained for polynomials over $\mathbb{Z}$.

I propose also to to look at designing parallel algorithms for linear algebra over algebraic number, algebraic function and rational function fields. For many operations (determinant, linear systems, characteristic polynomial) it will be natural to consider modular algorithms which are easily parallelized. For example, the modular algorithm of Chen and Monagan from [3] for solving linear systems over Cyclotomic fields could be parallelized.

Today multi-core computers are the norm but the processing capability of GPUs and hybrid facilities is becoming the new norm. Facing the GPU reality, I propose also to develop a fast bivariate GCD routine for a GPU.

1. Polynomial factorization over algebraic number and functions fields and 2. Parallel polynomial multiplication modulo triangular sets.

We consider the problem of factoring a multivariate polynomial $A(x_1, x_2, ..., x_n)$ over an algebraic number field $\mathbb{Q}(\alpha)$ or function field in k parameters $t_1, t_2, ..., t_k$. In general the number field (function field) may be presented with multiple field extensions whose minimal polynomials form a triangular set T. For example, for the function field $F = \mathbb{Q}(t)(\sqrt{2}, \sqrt{t})$, the triangular set T is $\{z_1^2 - 2, z_2^2 - t)\}$. We let D denote the degree of F (4 in this example).

Trager's method from [27] reduces factorization in $F[X]$ to $\mathbb{Q}[X]$. It increases the degree by a factor of D *in each variable* which introduces a severe expression swell when applied to multivariate polynomials. In [28], Wang searches for a prime p for which the minimal polynomial $m(z)$ for $\mathbb{Q}(\alpha)$ is irreducible. If such a prime exists (they don't in general) then factorization over $\mathbb{Q}(\alpha)$ is reduced to factorization over the finite field over $GF(p^D)$ with no expression swell. In [30] Zhi factors a univariate image in $F[x_1]$ and uses Hensel lifting over $\mathbb{Q}$ to recover $x_2, ..., x_n$. In [13], we also use Hensel lifting but modulo p and use $p-adic$ lifting to recover the rational coefficients. The Hensel lifting is done modulo T and hence the efficiency of factorization depends on the efficiency of polynomial multiplication modulo T modulo p. We use pseudo-division to avoid fractions in the parameters.

Currently, the best approach to factor multivariate polynomials over a field F is to factor a univariate image over F and recover the other variables, one at a time, using Hensel lifting. When lifting a variable y we recover the factors modulo $(y-\alpha)^{(k+1)}$ from the factors modulo $(y-\alpha)^k$. This whole process is sequential, however, as we recover variables one at a time, and each variable one degree at a time which limits parallelism..

To improve factorization over number fields and function fields we propose the following. I give details on each separately – especially 3 as it is not obvious how to do that.

1. Develop a parallel algorithm for polynomial multiplication over $\mathbb{Q}(\omega)$.
   In general parallelize multiplication modulo an arbitrary triangular set T.
2. Improve the complexity of diophantine equation solving for the multivariate case.
   It is a bottleneck of multivariate factorization.
3. Experiment with using interpolation to recover the factors instead of Hensel lifting to increase parallelism.

1: Multiplication in $\mathbb{Q}[z_1, ..., z_l][x_1, ..., x_n]$ modulo T could be parallelized on the coefficients or we could use an evaluation/interpolation approach for the variables and a modular approach for $\mathbb{Q}$ which generates a lot of parallelism. [For the algebraic function case one would also interpolate the parameters $t_1, ..., t_k$.] I don't know which approach will be best so I think we must experiment here. Asymptotically fast algorithms for arithmetic in $\mathbb{Z}_p[z_1, ..., z_l]$ modulo T are studied by Li, Moreno Maza and Schost in [17]. This helps for large D. This C library has been integrated into Maple (see Li et. al. [18]) as the `modpn` library. Also, our `recden` C library (see Javadi and Monagan [14]) for arithmetic with polynomials over algebraic number fields modulo p can be used.

2: Suppose $A(x, y) = B(x, y)C(x, y)$ in $F[x, y]$ and suppose the degree of both B and C is N in both x and y. Assuming classical $O(N^2)$ multiplication in $F[y]$ and $F[x]$, Miola and Yun [19] showed how to modify linear Hensel lifting to update the error $A - B^{(k)}C^{(k)} \bmod p^k$ efficiently to reduce the total cost of linear Hensel lifting from $O(N^5)$ to $O(N^4)$. The same construction can be applied to the multivariate case. However in the multivariate case, the solution of the diophantine equations that arise begins to dominate the cost. My student B. Tuncer is looking for a way to allow us to efficiently update a diophantine solution.

3: Suppose a polynomial $A(x, y, z)$ factors as $B(x, y, z) \times C(x, y, z)$. Suppose we factor three univariate images $A(x, \alpha_1, \beta_1)$, $A(x, \alpha_2, \beta_2)$ and $A(x, \alpha_3, \beta_3)$ over $\mathbb{Q}$ and use Hensel lifting on y to obtain

$$\begin{aligned} A(x, y, \beta_1) &= f_1(x, y, \beta_1) \times f_2(x, y, \beta_1) \\ A(x, y, \beta_2) &= g_1(x, y, \beta_2) \times g_2(x, y, \beta_2) \\ A(x, y, \beta_3) &= h_1(x, y, \beta_3) \times h_2(x, y, \beta_3) \end{aligned}$$

In general, we won't know if $B(x, y, \beta_1)$ is f_1 or f_2. If, however, the monomials in $B(x, y)$ and $C(x, y)$ are different then we can identify which factors here are the image of $B(x, y, z)$ and interpolate those together. If not, then our idea is the following. Apply Hensel lifting in z once to obtain $A(x, \alpha, z) = B(x, \alpha, z) \times C(x, \alpha, z)$. Then, with high probability $B(x, \alpha, \beta_1) = f_1(x, \alpha, \beta_1)$ or $f_2(x, \alpha, \beta_1)$ but not both. Similarly $B(x, \alpha, \beta_2) = g_1(x, \alpha, \beta_1)$ or $B(x, \alpha, \beta_2) = g_2(x, \alpha, \beta_1)$ but not both. Thus using one additional bivariate factorization enables us to interpolate z. We get a dense interpolation method from this which enables us to parallelize the algorithm. We seek a sparse interpolation method. Related work includes the Black Box model of Kaltofen and Trager in [15] and Diaz and Kaltofen's FoxBox implementation in [5].

It is very helpful to have good application problems to test algorithms with. Randomly generated problems are not always good problems to use. Indeed one of my goals is to build a database of real problems. One good problem is the following. Let $V_n = [x_1, x_2, ..., x_n]$. Let C_n be the n by n matrix whose rows are formed from V_n by shifting the variables cyclically. For example

$$C_4 = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ x_2 & x_3 & x_4 & x_1 \\ x_3 & x_4 & x_1 & x_2 \\ x_4 & x_1 & x_2 & x_3 \end{bmatrix}$$

Let $D_n = \det(C_n)$. So D_n is a polynomial in $\mathbb{Z}[x_1, x_2, ..., x_n]$. It is known (see Kra and Simanca [25]) that D_n factors over $\mathbb{Q}(\omega)$ into n linear factors where ω is a primitive n'th root of unity. The computational problem is to compute and factor D_n over $\mathbb{Q}$ and $\mathbb{Q}(\omega)$ for as large an n as is possible. Using our fast multiplication in [23] we can compute D_{14} in 125s but factoring it is much harder. Table 1 gives some timing data showing where we are at.

n	D_n		$\mathbb{Q}$	$\mathbb{Q}(\omega)$
12	2.6s	86500 terms	32.7s	60.7s
13	15.5s	400024 terms	8.5m	> 10hrs

Table 1: CPU times for computing D_n and factoring D_n over $\mathbb{Q}$ and $\mathbb{Q}(\omega)$ in Maple 17

3. Multivariate polynomial interpolation and
4. Parallel algorithms for polynomial GCD and Linear Algebra.

In current work, my student Matthew Gibson and I have developed a parallel library for polynomial GCD computation in $\mathbb{Z}[x, y, ...]$. We have parallelized Brown's GCD algorithm [2] using Cilk. Our software maps to multiple bivariate GCDs in $\mathbb{Z}_p[x, y]$ which are computed in parallel. We use machine 63 bits on a 64 bit machine. We have optimized the serial bivariate GCD code achieving a speedup of a factor of 10 over Maple and Magma. For $\mathbb{Z}[x, y]$ we are seeing a parallel speedup on of a factor of 15 on a 16 multi-core server for an overall improvement of a factor of 150 over Maple and Magma. We would like to try a GPU implementation as this could lead to a further 10-fold improvement. For this we would need to implement univariate evaluation, interpolation and the Euclidean algorithm on a GPU.

Recent work on this includes the work of [12], Haque and Moreno Maza (2012) who experimented with a parallel Euclidean algorithm for $\mathbb{Z}_p[x]$ on a GPU. They showed that quadratic $O(d^2)$ algorithms for polynomial multiplication and division and in $\mathbb{Z}_p[x]$ were faster than FFT based algorithms on a GPU up to degee 2^{12} and for GCD up to degree 2^{18}. We do not plan to parallelize GCD in $\mathbb{Z}_p[x]$ but rather run multiple instances of $\mathbb{Z}_p[x]$ in parallel. Related work includes that of Emeliyanenko [8] (2013) and Stussak and Schenzel [26] (2012) for computing resultants in $\mathbb{Z}[x, y]$ on a GPU. Emeliyanenko used a matrix based approach. Stussak and Schenzel use the Euclidean algorithm which should be an order of magnitude faster.

Structured Polynomial Interpolation

A long standing problem is Computer Algebra has been "How fast can we interpolate a sparse polynomial in N variables of total degree D with T non-zero terms with integer coefficients?" Related work includes the work done by Grigoriev and Lakshamn (1995) [11] and Giesbrecht, Kaltofen and Lee (2003) [9], and others to determine the *sparsest shift* of a univariate polynomial. That is, find α such that $f(x - \alpha)$ is sparse, and if such α exists, find the coefficients of $f(x - \alpha)$. Here, motivated by applications requiring interpolation, I propose to work on what I call "structured" polynomial interpolation. I will explain.

In the summer of 2013 Manuel Kauers sent me a sequence of $n \times n$ matrices A in two parameters w and z arising from a combinatorial setting. He sent me matrices for $n = 32, 64, 128, 256$. He wanted to compute and factor the characteristic polynomial $C(x, w, z) = x^n + \ldots$ of $A(w, z)$. We were able to get Maple and Sage to compute the $n = 32$ case fairly quickly using a classical approach (the division free approach of Berkowitz) but unable to compute the $n = 64$ case in either system. I eventually did succeed using Maple on a machine with more memory – it took 7 days.

It seemed to us that a modular approach should be faster and also more easily parallelized. The approach would be to compute images $C(x, w = \alpha_i, z = \beta_j)$ modulo primes p_k, recover $C(x, w, z)$ by interpolating w and z and applying Chinese remaindering. Each image $C(x, \alpha_i, \beta_j) \bmod p_k$ can be computed in $O(n^3)$ arithmetic operations in $\mathbb{Z}_p$ using e.g. the Hessenberg method (see Ch. 2 of Cohen [4]). Bounds for $\deg_w(C)$ and $\deg_z(C)$ can easily be determined from $A(w, z)$ by inspection. A bound for the largest integer coefficient of C can also be determined from $A(w, z)$ using Goldstein and Graham [10].

However, we noticed that each coefficient $f_i(w, z)$ of x^i of the characteristic polynomial has a lot of structure which, if intelligently exploited, can be used to greatly reduce the work that our modular algorithm does. The coefficients $f_i(w,z)$ are of the form

$$f_i(w,z) = w^{L_i} z^{N_i} (z^4-1)^{M_i} g_i(w^2,z^4)$$

for non-negative integers L_i, N_i and M_i. Some of this information e.g. that $C(x,w,z)$ has even powers in w and z can be obtained by inspecting the input matrix A. The rest we can "discover" by picking a random point $\alpha \in \mathbb{Z}_p$, interpolating $C(x,\alpha,z)$ and $C(x,w,\alpha)$ from points [this costs $O(n^4)$], then inspecting the coefficients of $C(x,\alpha,z)$ and $C(x,y,\alpha)$. Here is some data for $n = 64$.

i	L_i	M_i	N_i	$\deg_w(g_i)$	$deg_z(g_i)$
0	384	384	193	0	0
1	372	384	187	6	3
32	132	120	67	60	66

Since $\deg_w(C) = 384$, $\deg_z(C) = 1152$, $\max_{i=0}^{64} \deg_w(g_i) = 60$ and $\max_{i=0}^{64} \deg_z(g_i) = 66$, the total reduction in the cost to interpolate w and z is a factor of $384/60 \times 1152/66 = 111$. By removing the factors of z^4-1 we also reduce the length of the largest integer coefficient of $C(x,w,z)$ by a factor of 2.6 for a total reduction of a factor of 289. The time for my modular implementation to compute $C(x,w,z)$ is 49.1 seconds CPU and 15.1 seconds real time using 4 cores (we needed 4 primes).

The goal of this project is to try to automate our observations and build software that "intelligently" interpolates coefficients of polynomials. We need to

- Decide what structure we can afford to look for.
- Analyze the overall probability of error in determining L_i, N_i, M_i.
- Design how the algorithm will detect an incorrect assumption – at present we don't.
- Design how to parallelize the algorithm – we presently parallelize at the level of primes and implement the parallelization at the Unix command level by calling Maple (Sage) on multiple inputs and communicating data via files.

I think this is a very good thesis topic for a student. There is some design, parallel programming, and analysis involved. This idea can potentially be applied to many multivariate interpolation problems, including determinants, resultants, rational function solutions of linear systems, and polynomial GCD problems.

Impact

This research proposal covers core facilities in Computer Algebra Systems. It will find many practical applications in science and engineering which involve multivariate polynomials, algebraic numbers and matrices with parameters.

The focus on parallel algorithms, their design and implementation, will be of benefit to students. For those who leave the field of Computer Algebra, the algorithm design and parallel programming skills learned will be marketable.

References

[1] M. Ben-Or and P. Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proceedings of STOC '88*, ACM Press, pages 301–309. 1988.

[2] W. S. Brown. On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors. *J. ACM*, **18** (4), 478–504, 1971.

[3] L. Chen, M. Monagan. Solving Linear Systems of Equations Over Cyclotomic Field. *J. Symb. Cmpt.* **45** (9) 902–917, 2010.

[4] H. Cohen. *A Course in Computational Algebraic Number Theory.* Graduate Texts in Mathematics, Springer Verlag, 1993.

[5] A. Diaz and E. Kaltofen. FOXBOX: a system for manipulating symbolic objects in black box representation. *Proceedings of ISSAC '98*, ACM Press, pp. 30–37, 1998.

[6] Jean-Guillaume Dumas, Thierry Gautier et Jean-Louis Roch. Generic design of Chinese remaindering schemes. *Proceedings of PASCO '10*, pp. 26–34, ACM Press, 2010.

[7] J.G. Dumas, T. Gauthier, M. Giesbrecht, P. Giorgi, B. Hovenin, E. Kaltofen, D. Saunders, W.J. Turner and G. Villard. LinBox: A Generic Library for Exact Linear Algebra. Research Report N^o 2002-15, INRIA, Lyon, 2002.

[8] Emeliyanenko, P. Computing resultants on Graphics Processing Units: Towards GPU-accelerated computer algebra. *J. Parallel and Dist. Cmpt* **73**(11) 1495–1505, 2013.

[9] M. Giesbrecht, E. Kaltofen, W. Lee. Algorithms for Computing Sparsest Shifts of Polynomials in Power, Chebyshev, and Pochhammer Bases. *J. Symb. Cmpt.* **36**(3–4) 401–424, 2003.

[10] A. J. Goldstein and R. L. Graham. A Hadamard-type bound on the coefficients of a determinant of polynomials. *SIAM Review* **1** 394–395, 1974.

[11] D. Y. Grigoriev, Y. N. Lakshman, Algorithms for computing sparse shifts for multivariate polynomials. *Proceedings of ISSAC 95*, ACM Press, pp. 96-103, 1995.

[12] S.A. Haque, M. Moreno Maza. Plain Polynomial Arithmetic on GPU. *Proceedings of HPCS 2012*, *J. Physics: Conference Series* **385** pp. 1–10, 2012.

[13] Mahdi Javadi and Michael Monagan. On Factorization of Multivariate Polynomials over Algebraic Number and Function Fields. *Proceedings of ISSAC '09*, pp. 199–206, ACM Press, 2009.

[14] Mahdi Javadi and Michael Monagan. In-place Arithmetic for Univariate Polynomials over an Algebraic Number Field. *Proceedings of ASCM 2009*, Math-for-Industry Lecture Notes Series, **22**, Kyushu University, pp. 330–341, 2009.

[15] E. Kaltofen and B. Trager. Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators. *J. Symb. Cmpt.*, **9**, pp. 301–320, 1990.

[16] E. Kaltofen, W. Lee, and A. A. Lobo. Early termination in Ben-Or/Tiwari sparse interpolation and a hybrid of Zippel's algorithm. *Proceedings of ISSAC '00*, ACM Press, pp. 192–201, 2000.

[17] Xin Li, Marc Moreno Maza and Éric Schost. Fast arithmetic in triangular sets: From theory to practice. *J. Symb. Cmpt.*, **44**(7): 891-907, 2009.

[18] Xin Li, Marc Moreno Maza, Raqeeb Rasheed, and Éric Schost. The `modpn` library: Bringing fast polynomial arithmetic into MAPLE. *J. Symb. Cmpt.* **46**(7): 841-858, 2011.

[19] A. Miola, D. Y. Y. Yun, Computational Aspects of Hensel-type Univariate Polynomial Greatest Common Divisor Algorithms. *Proc. EUROSAM '74* (1974), pp. 46–54.

[20] M. B. Monagan. Probabilistic Algorithms for Resultants. *Proceedings of ISSAC '2005*, ACM Press, pp. 245–252, 2005.

[21] Michael Monagan and Roman Pearce. POLY: A new polynomial data structure for Maple 17. To appear in the post conference proceedings of ASCM 2012, Springer Verlag, 2014. Preprint: http://cecm.sfu.ca/CAG/papers/ascmMUL19.pdf

[22] Michael Monagan and Roman Pearce. POLY: A new polynomial data structure for Maple 17. *Communications in Computer Algebra* **46**(4), 164−167, 2012.

[23] Michael Monagan and Roman Pearce. Parallel Sparse Polyomial Multiplication using Heaps. *Proceedings of ISSAC '09*, ACM Press, pp. 263–269, 2009.

[24] Roman Pearce and Michael Monagan. Parallel Sparse Polynomial Division using Heaps. *Proceedings of PASCO '2010*, ACM Press, pp. 105–111, 2010.

[25] Irwin Cra and Santiago R. Simanca. On Circulant Matrices. *Notices of the American Math. Soc.* **59**(3) 368–377, March 2012.

[26] C. Stussak and P. Schenzel. Parallel computation of bivariate polynomial resultants on Graphics Processing Units. *Proceedings of Applied Parallel and Scientific Computing, 2010*, Springer Verlang LNCS **7134**, pp. 78–87, 2012.

[27] B. Trager. Algebraic Factoring and Rational Function Integration. *Proceedings of ISSAC '76*, ACM Press, pp 219–226, 1976.

[28] P. S. Wang. Factoring multivariate polynomials over algebraic number fields. *Math. Cmpt.* **30** 324–336, 1976.

[29] R. Zippel. Probabilistic Algorithms for Sparse Polynomials. *Proceedings of Eurosam 79'*, Springer-Verlag LNCS, **72**, 216-226, 1979.

[30] Lihong Zhi, Algebraic Factorization and GCD Computation. *Mathematics Mechanization and Applications,* Academic Press, 325–342, 2000.