Form 101 Part II: Research Proposal

Synopsis

Provide a concise overview of the scientific and technical objectives, approach, and the new knowledge, expertise, or technology that could be transferred to Canadian industry. Indicate the benefits expected to accrue to Canadian industry, to the academic institution, and to the scientific or engineering discipline.

The industrial partner for this project is Waterloo Maple Inc., a Canadian based company owned by Cybernet Systems Inc. of Japan whose main commercial product is Maple. Maple is a mathematical software system with tools for exact algebraic computation and numerical computation. It is used by scientists and engineers in Canada, the US, and worldwide, for research and development. Our university has a campus wide site license for Maple. Many members of our department of mathematics use Maple for their teaching and research.

Maple's main algebraic capability is polynomial computation over various rings, most importantly, the integers, finite fields, algebraic number fields, and function fields. Most user level application facilities in Maple, such as solving systems of algebraic and differential equations, make extensive use of polynomial operations. These include (i) polynomial multiplication and division, (ii) polynomial greatest common divisors (GCDs), (iii) polynomial factorization over various fields, and (iv) Gröbner basis computation.

Perhaps the biggest single challenge facing the developers of computer algebra systems like Maple is whether we can retool them to take full advantage of multi-core processors.

In previous work we improved Maple's speed for multiplication and division of multivariate polynomials with integer coefficients by a factor of more than 200 (over Maple 13) on a quad-core desktop (see [24, 27]). We achieved this by developing improved heap based algorithms in [24] and then by parallelizing them in [25, 26]. This work was integrated into Maple 14 and Maple 16. This means some parts of Maple can handle much larger polynomials. But it also means that Maple now multiplies and divides more than 200 times faster than it computes GCDs and so GCD computation is now a bottleneck.

The main objectives of this proposal are (i) to design and implement new efficient algorithms for polynomial GCD computation which can be parallelized, (ii) to build a new 63 bit software library for polynomial operations in $\mathbb{Z}_n[x]$ for implementing modular algorithms, (iii) to design and implement, in C, parallel algorithms for polynomial operations for multi-core computers, and (iv) to begin exploiting GPUs as Intel and AMD are now moving away from multi-core CPUs to hybrid chips with CPU and GPU capabilities.

If successful, this will make many of Maple's core algebraic facilities the fastest available for general purpose algebraic computation, and much faster than Maple's main commercial competitor, `Mathematica`. Because the focus of this proposal is the "core" algebraic facilities, this project will benefit many Maple users and it will provide Maple with a solid platform for future research and development in computer algebra. The main research contribution will be the development and implementation of efficient algorithms for computing with polynomials on multi-core platforms.

Background

Relate the proposal to current scientific, technical and commercial developments in the field, referring to the current literature and market conditions. Describe the background research on which the project is built.

The four main problems addressed in this project proposal are

1. Polynomial GCD computation and sparse polynomial interpolation.
2. Software libraries for arithmetic in $\mathbb{Z}_p[x]$ for implementing modular algorithms.
3. Exact sparse linear algebra and Gröbner basis computation.
4. Data structures for multivariate polynomials with parallel speedup.

1: The polynomial GCD problem.

The first effective algorithm for *multivariate* polynomial GCD computation over $\mathbb{Z}$ was Brown's algorithm [1] in 1971. Subsequent research focused on improving the complexity of Brown's algorithm for *sparse* polynomials because multivariate polynomials occurring in applications are mostly sparse. Some key results include Zippel's SMGCD algorithm [2] from 1979 which uses sparse interpolation, Wang's EEZ-GCD algorithm [3] from 1980 which uses Hensel lifting, and Kaltofen and Trager's black box GCD algorithm [4] from 1990. Zippel's algorithm is currently used in Mathematica, Maple and Magma. Maple uses our version of Zippel's algorithm from 2005 (see [5]) which solves the leading coefficient problem.

Although Brown's algorithm is naturally parallel, the sparse methods are not. Hensel lifting is sequential; one recovers the coefficients of the GCD one variable at a time and for each variable, one degree at a time. Zippel's algorithm also recovers the coefficients of the gcd one variable at a time. However, the sparse interpolations needed for each variable can be done in parallel. In 1994 Rayes, Wang and Weber in [6] parallelized this and other parts of Zippel's algorithm with some success. As far as we know, no parallel work has been done since then and no computer algebra system has a parallel sparse GCD implementation.

Subsequent research in polynomial GCD computation focused mainly on number fields. For $\mathbb{Q}(\alpha)[x]$, Maple and Magma use Encarnacion's algorithm ([7], 1995) which uses *rational number reconstruction*, a tool developed by Wang, Guy and Davenport in [8]. For $\mathbb{Q}(\alpha_1, \alpha_2, ..., \alpha_k)[x_1, x_2, ..., x_n]$, Maple uses our generalization of Encarnacion's algorithm from [45]. For polynomials over an algebraic function field van Hoeij and Monagan ([9], 2004) showed how to interpolate the polynomial variables and parameters one at a time using univariate *rational function reconstruction*. But this is a dense method. In 2007, [10] Monagan and Javadi modified the algorithm to use Zippel's sparse interpolation and also to handle multiple field extensions. To implement our algorithms we developed a *recursive dense* polynomial data structure (see [45, 11]) that supports multiple algebraic extensions over $\mathbb{Q}$ and $\mathbb{Z}_p$. This work was integrated into Maple 14. However, there is no parallel support.

Zippel's sparse interpolation [2] is central in several of the GCD algorithms mentioned. It interpolates a polynomial in n variables with t terms of degree d using $O(ndt)$ points. The interpolations are done modulo a prime p and require $p \gg dt$ to work with high probability. In [12] 1988, Ben-Or and Tiwari devised a deterministic sparse interpolation algorithm for characteristic 0 that requires only $O(t)$ points. One evaluates at $(2^i, 3^i, 5^i, ..., p_n^i)$ for $i =$

$0, 1, ..., 2T-1$ where p_n is the n'th prime for a term bound $T \geq t$. To use their algorithm modulo p requires $p > p_n^d$ which can be large. E.g. for 10 variables and $d = 100$, we would need $p > 29^{100}$, a 147 digit prime which would make the modular arithmetic very slow.

Several authors have modified the Ben-Or/Tiwari algorithm to work over smaller finite fields. In [13] 2000, Kaltofen, Lee, and Lobo replace the dense interpolation step in Zippel's algorithm with Ben-Or/Tiwari interpolation. Their algorithm requires $O(nt)$ points and can be used with small primes. However, the modification serializes the interpolation of each variable. In [14] 2010, Monagan and Javadi modify the Ben-Or/Tiwari algorithm to interpolate the variables in the monomials in parallel, also using $O(nt)$ points. As far as we know, no one has tried Ben-Or/Tiwari interpolation for polynomial GCD computation.

2: Software libraries for arithmetic in $\mathbb{Z}_p[x]$.

Algorithms for computing GCDs and resultants of polynomials with integer coefficients use *modular algorithms*. One computes the GCD (resultant) modulo a prime p. For efficiency one uses machine primes so that arithmetic in $\mathbb{Z}_p$ can be done by the hardware of the machine. One reduces to a univariate problem in $\mathbb{Z}_p[x]$ by evaluation. One needs to implement polynomial multiplication, division with remainder, and the Euclidean algorithm. Asymptotically fast algorithms for these operations are well known (see von zur Gathen and Gerhard [15]).

Each computer algebra system has its own library for $\mathbb{Z}_p[x]$. This library is also used for factoring polynomials with integer coefficients. In Maple it is the `modp1` library which we developed in 1992 (see [16]). It uses machine arithmetic for primes less than $\sqrt{2^{31}}$ on a 32 bit machine and $\sqrt{2^{63}}$ on a 64 bit machine. It has Karatsuba multiplication but no FFT.

The best open source library for $\mathbb{Z}_p[x]$ is Victor Shoup's C++ library NTL (see [17]). Though the just released FLINT 2.3 ([19]) claims to be competitive. Both Magma and NTL support the FFT and Magma has fast gcd. Because of this, Magma is faster than Maple for polynomials with high degree. A new experimental library is David Harvey's C library see [18]. It supports 63 bit primes and has an FFT for multiplication but no fast gcd. We are also developing a new C library for 63 bit primes with an FFT for multiplication. Our student Soo Go has implemented the fast Euclidean algorithm. A drawback of all these $\mathbb{Z}_p[x]$ libraries is that none have multi-core or GPU support.

Related work includes Maple's `modpn` library which uses the FFT extensively. It was developed by Xin, Moreno Maza, Rasheed and Schost in 2009 (see [21]) for fast arithmetic modulo triangular sets over $\mathbb{Z}_p$. Some work has also been done for multi-core computers (see [22]) and for GPUs (see [23]).

3: Gröbner basis computation.

Many of the early algorithmic developments in Gröbner basis computation were variants of Buchberger's original algorithm that reduced the number of S-polynomial pairs considered. We mention Buchberger's "normal" strategy [30], and the "sugar" strategy of Traverso et. al. [31]. The bottleneck in these algorithms is division of S-polynomials by the partial basis. These divisions were expensive and often produced zero, i.e., no new information.

In 1999 [32], Faugère used sparse linear algebra to simultaneously reduce polynomials in batches. This produced a two orders of magnitude speedup because (i) the cost of searching for a divisor for each monomial was amortized over many polynomials, and (ii) the division

algorithm for even a single polynomial had been inefficient. Sequential implementations of Faugère's algorithm, known as F4, are used in Magma and Maple.

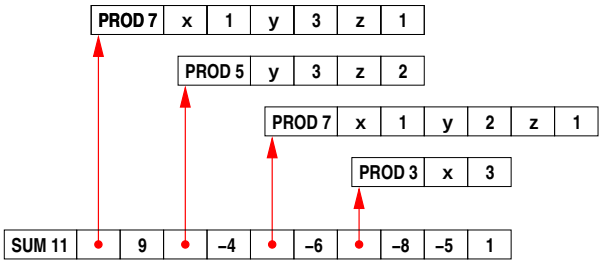
The problem with the division algorithm was recognized by Yan [33], and starting from much earlier work of Johnson [34], we created an algorithm with lower arithmetic, sorting, and storage costs in [24], and a parallel version with superlinear speedup in [26]. In [35] 2012, Roune and Stillman tried ours and other data structures and showed that the improvements to division close the gap between division based methods and F4.

Faugère went on to develop the F5 algorithm in [36], which reduces the amount of work in Gröbner basis computations by the use of a *signature scheme*. The criteria permitted by these schemes has been the subject of intense investigation, recently culminating in [35].

In [38], Faugère and Sylvain describe one way to parallelize the sparse linear algebra in F4/F5, which consists of direct elimination of large structured sparse systems. Faugère's new implementation of F5 is much faster than existing Gröbner basis routines, beating Magma by a factor of 88 on 8 cores on one benchmark. This year has seen the first workshop devoted exclusively to this problem: "Efficient Linear Algebra for Gröbner Basis Computations" at Kaiserslautern (see `http://wiki.lmona.de/events/elagb`).

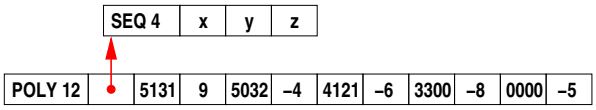
4: Multivariate polynomial data structures.

Arguably, the most important data structure in a computer algebra system is the one used for multivariate polynomials. Maple, Magma, Mathematica and Singular all use a sparse distributed representation in which only non-zero terms are stored. For example, on the left of the figure below is Maple's current data structure. It is a sum-of-products. Each monomial is stored as a separate object labelled a `PROD`. The data structures in Magma, Mathematica and Singular are similar; they all contain pointers to monomials or terms. But these are not suitable for modern processors which need sequential memory access for maximum performance. Moreover, monomial multiplications are expensive; each requires allocation of memory and a loop. This slows down polynomial multiplication.



Maple's sum-of-products representation has irregular Maple dags for each term.

Representations for the polynomial $9\,xy^3z - 4\,y^3z^2 - 6\,xy^2z - 8\,x^3 - 5$.



Our packed distributed representation uses bit fields and sorts by total degree.

The figure on the right shows our new data structure for polynomials. The second word is a pointer to the variables. This is followed by monomials and coefficients where the monomials encode the total degree and the exponents in a single machine word. E.g. for xy^3z we store the total degree 5 and the exponents $(1,3,1)$ as $5\cdot 2^{48} + 2^{32} + 3\cdot 2^{16} + 1$ on a 64 bit machine. This encoding means the terms can be sorted into graded lex ordering by comparing monomials as unsigned integers. Monomial multiplications are done by integer addition which also only costs one machine instruction!

Initially we used this data structure to implement our heap based multiplication and division algorithms in [24, 25, 26]. When we integrated our software into Maple (see [27]), we also saw a huge speedup in multivariate polynomial factorization (compare Maple 16 with Maple 13 in Table 1). This is because most of the time in factoring polynomials is in Hensel lifting and most of the time in Hensel lifting is in polynomial multiplication.

	Maple 13	Maple 16		new POLY dag		Magma	Mathem-
benchmark		1 core	4 cores	1 core	4 cores	2.16-8	atica 7.0
#1	31.10	2.66	2.54	1.06	0.93	6.15	11.82
#3	391.44	15.70	13.47	8.22	6.13	117.53	164.50
#4	2953.54	56.68	44.06	26.43	16.17	332.86	655.49

Table 1: Timings (in seconds) for three multivariate factorizations from [28].

But there is little parallel speedup. We noticed that conversion overhead was often over 50% of the multiplication time which ruins parallel speedup. In current work [28], we have tried making the new POLY data structure the default in the Maple kernel to eliminate conversions. To do this, we coded many Maple kernel operations. Some are now very fast. The results (see new POLY dag timings in Table 1) are very promising. We are now 13x, 27x and 40x faster than Mathematica, respectively.

How much faster can we factor these multiviarate polynomials on a multi-core computer? What about factoring polynomials over algebraic number fields and function fields? The multivariate algorithm of Javadi and Monagan in [29] also uses Hensel lifting. It uses repeated pseudo-division to reduce modulo a triangular set. We think we should be able to achieve similar speedups there as well by using better data structures, improving the division algorithm, and adding multi-core support.

Detailed Proposal

Discuss the scientific issues, research problems or technical complexities, and describe the research methodology and experimental design proposed to explain or resolve them. Provide a workplan and relate it to the milestone schedule. Describe the roles of any undergraduate or graduate students, or postdoctoral fellows who will be involved in the project.

1: Polynomial GCD sub-project.

We propose to develop a new sparse GCD algorithm based on a modified Ben-Or Tiwari interpolation for polynomials with integer coefficients and attempt a parallel implementation for multi-core platforms.

Let f_1, f_2 be polynomials in $n+1$ variables and let $g = gcd(f_1, f_2)$ be their GCD. Let t be the number of terms of g and d be the total degree of g. To compute g modulo a prime p, Zippel's algorithm uses $O(ndt)$ points and the algorithms of Kaltofen, Lee, and Lobo in [13], and Monagan and Javadi in [14] both use $O(nt)$ points. We propose here to develop a deterministic algorithm that would require only $O(t)$ points, thus optimal up to a constant factor and hence good for dense polynomials too. We outline the method here.

We pick relatively prime integers $q_1, ..., q_n$ satisfying $q_i > d$ until $p = q_1 q_2 \cdots q_n + 1$ is prime. This is easy to do. This choice means that we can now compute a discrete logarithm

in $\mathbb{Z}_p$ efficiently (in time no worse than $O(n\sqrt{d})$ arithmetic operations in $\mathbb{Z}_p$ using the Pohlig-Helman algorithm [39]). We pick a primitive element $\alpha \in \mathbb{Z}_p$ (again, easy to do since we have a factorization for $p-1$) and let $\omega_k := \alpha^{p-1/q_k}$ so that ω_k is a primitive q_k'th root of unity. The idea is to replace the $(2^i, 3^i, 5^i, \ldots, p_n^i)$ in Ben-Or Tiwari with $(\omega_1^i, \omega_2^i, \ldots, \omega_n^i)$.

Suppose $M = x_1^{e_1} x_2^{e_2} \cdots x_n^{e_n}$ is a monomial in g and $m = \omega_1^{e_1}\omega_2^{e_2} \times ... \times \omega_n^{e_n}$ is its value, obtained in the Ben-Or Tiwari algorithm. Then the discrete logarithm of m satisfies

$$\log_\alpha m \equiv e_1 \log_\alpha w_1 + ... + e_n \log_\alpha w_n \equiv e_1 \tfrac{p-1}{q_1} + ... + e_n \tfrac{p-1}{q_n} \pmod{p-1}.$$

Since the q_k are relatively prime and divide $p-1$, the exponents $e_1, e_2, ..., e_n$ can be obtained by computing $y := \log_\alpha m$ then solving $y \equiv e_k \frac{p-1}{q_k} \pmod{q_k}$ for e_k. The advantages of this approach are that we reduce the number of points needed to $O(t)$ and we can compute

$$g_i := \gcd(f_1(x_0, \omega_1^i, \ldots, \omega_n^i),\ f_2(x_0, \omega_1^i, \ldots, \omega_n^i)) \quad \text{for } i = 0, 1, 2, \ldots, 2T-1$$

in parallel. The disadvantages are several and these problems need efficient solutions.

First, the method requires p to be substantially larger than for Zippel's method. E.g., for $n = 10$ variables and degree $d = 50$, we will need $p > 50^9$, a 51 bit prime whereas Zippel's method could use a 31 bit prime (or even smaller). The Maple library for arithmetic in $\mathbb{Z}_p[x]$ only supports 31.5 bit primes on a 64 bit machine. Larger primes are done using software arithmetic which is slow. We address this problem in sub-project 2 below.

Second, whereas it is easy to get good degree bounds for g, good term bounds for t are not available. One must run the algorithm for $T = 2, 4, 8, 16, 32, ...$ until we "know" t. This serializes that part of the algorithm and complicates the parallel implementation.

Third, we cannot solve the "leading coefficient problem" using our univariate methods in [5] since Ben-Or/Tiwari interpolates all variables together. Let $g = a_d x_0^d + ... + a_1 x_0 + a_0$ be the gcd of f_1 and f_2. To interpolate g from monic $g_i \in \mathbb{Z}_p[x_0]$ we must attach $a_d(\omega_1^i, \ldots, \omega_n^i)$ (the image of the leading coefficient) to g_i. But we do not know $a_d(x_1, \ldots, x_n)$. Zippel's own implementation in Macsyma computes Δ the gcd of the leading coefficients of f_1 and f_2 and uses $\Delta(\omega_1^i, \ldots, \omega_n^i)$ instead. But Δ can be a non-trivial multiple of a_d. And factoring Δ and identifying which factors of Δ are in a_d using Wang's coefficient determination algorithm (see [40]) will limit parallel speedup. This problem needs a new solution.

Remark: We got the idea for the method presented here after studying the algorithm for interpolating polynomials with *numerical* coefficients of Giesbrecht, Labahn and Lee in [41] and through correspondence with Kaltofen. Giesbrecht told us he was also aware of the method and that Kaltofen was aware of it in 1988 but did not realize at that time that the discrete logarithms could be computed efficiently.

Remark: most of the time in the new GCD algorithm will often be evaluating the input polynomials f_1 and f_2 at $(\omega_1^i, \ldots, \omega_n^i)$ modulo p. This task appears to be ideal for a GPU. Recently, Vershelde and Yoffe in [44] looked at this problem for floating point evaluations.

This project is suitable for a PhD student interested in mathematics and computing. A prototype sequential implementation in Maple would be made by the student in the first year of the project. A prototype parallel implementation using Cilk could then be attempted in the second year. The student will need to do some experimentation, code optimization and algorithm analysis.

We propose to use Cilk for this and the other sub-projects because we find it easy to parallelize existing C code using Cilk. To integrate parallel software into the Maple kernel, which is programmed in C, we would need to either develop our own parallel support infrastructure for C, or, simulate Cilk's model of parallel programming in Maple. We propose, together with Maplesoft personnel, to develop a prototype Maple kernel which supports the Cilk parallel primitives `cilk, spawn` and `sync`. If successful this will make it easier to integrate Cilk code into Maple.

2: $\mathbb{Z}_p[x]$ library sub-project.

We propose to develop a C library for polynomial arithmetic for $\mathbb{Z}_p[x]$ in the spirit of the GMP library [42] that has been developed for integer arithmetic. Our first goal is to support 63 bit primes on a 64 bit machine to support sub-project 1. We will also implement known asymptotically fast algorithms which will improve Maple's performance for polynomial factorization over $\mathbb{Z}_p[x]$ and hence also $\mathbb{Z}[x]$. Our second goal is support for multi-core platforms. We will attempt to parallelize polynomial multiplication, division, gcd and root finding for $\mathbb{Z}_p[x]$. In the space available we outline one approach for fast root finding. Our goal is to be 50 times faster than Magma on a quad-core desktop.

Let $M(d)$ be the cost of multiplying two polynomials in $\mathbb{Z}_p[x]$ of degree d. To compute the roots of the generating polynomial $\Lambda(x) = x^d + \ldots + \lambda_1 x + \lambda_0$ over a finite field $\mathbb{Z}_p$ (here $\Lambda(x)$ is known to be a product of linear factors), one computes

$$g(x) \quad = \quad \gcd(\left[(x+\alpha)^{(p-1)/2} \mod \Lambda(x)\right] - 1, \Lambda(x)) \tag{1}$$

for some α chosen randomly from $\mathbb{Z}_p$. This computation can be reduced to polynomial multiplication (see [15]) with asymptotic cost $O(M(d)\log p)$ for computing the power modulo $\Lambda(x)$ and and $O(M(d)\log d)$ for computing the gcd. The gcd splits $\Lambda(x)$ into $g(x)$ and $\Lambda(x)/g(x)$ for which we compute the roots recursively. The total cost for the root finding is $O(M(d)\log d \log p + M(d)\log^2 d)$ arithmetic operations in $\mathbb{Z}_p$.

In [43] we developed our own library for $\mathbb{Z}_p[x]$ for 31.5 bit primes. We implemented an FFT based multiplication and fast division. To compute the power $(x+\alpha)^{(p-1)/2} \mod \Lambda(x)$, we tried the following approach. Instead of reducing the computation to a sequence of multiplications which are computed with the FFT, we instead stay inside FFT co-ordinates where multiplication is linear, but also other operations are linear. Theoretically we reduced the number of FFTs by a factor of 2. With this and other improvements, we are 6× faster than Magma for computing all roots of polynomials in $\mathbb{Z}_p[x]$ for 31.5 bit primes.

To parallelize the root finding, after we have split $\Lambda(x)$ we can compute the roots of $g(x)$ and $\Lambda(x)/g(x)$ in parallel, and as we keep splitting, there is "as much parallelism as you want" according to von zur Gathen. However this doesn't work because the first power and gcd becomes a sequential bottleneck. So we must also parallelize the powering and gcd. For large d, some parallelism is available in the fast Euclidean algorithm – a factor of 8 is available in the matrix-matrix and two matrix-vector multiplications of polynomials.

This project is suitable for a PhD student and/or PDF interested in asymptotically fast algorithms and parallel algorithms. In the first year the students and researchers would implement 63 bit prime support for $\mathbb{Z}_p[x]$ in C. In the second year we would look at parallel speedup. For this the student would do a parallel implementation in Cilk.

We also want to support $\mathbb{F}_q[x]$ for small finite fields $\mathbb{F}_q$. Currently Maple has no special support for small finite fields or finite fields of characteristic 2. Arithmetic in $\mathbb{F}_q$ is done using polynomials over $\mathbb{Z}_p$ which is slow for small q. As an application of the library for $\mathbb{F}_q[x]$ we will develop a module for Coding Theory (algebraic error correcting codes). Such codes are nowadays critical in Engineering in the design of devices that communicate over noisy channels such as Internet or wireless networks. Cyclic codes, the most important family of algebraic codes, are encoded and decoded as polynomials over finite fields. (This includes BCH codes, Reed-Solomon codes, as well as other important families of codes.) This functionality has never been available in Maple whereas it is implemented extensively in its competitors such as Magma.

This project is suitable for a Masters student interested in coding theory, finite fields and computing. Maplesoft will participate in the design for the representation of elements of $\mathbb{F}_q$.

3: Linear algebra for Gröbner bases sub-project.

Maplesoft, our industrial partner, has asked us to help them develop a parallel C library for solving structured sparse linear systems to be used in Maple's Gröbner basis package. Our interest in sparse linear algebra is broader than this, and we think it could be a powerful tool for handling large scale algebraic problems. Maplesoft, also has an interest in the simplification of sparse algebraic models with tens of thousands of variables. Sparse linear algebra should be useful there.

We propose to solve the Gröbner basis problem as follows. Given a batch of S-polynomials in the F4 or F5 algorithms, we shall first divide each one by the basis using a heap. This is easily done in parallel, but we will need a cache to decide in $O(1)$ time which basis elements' leading monomials divide a given monomial. We will then use sparse linear algebra on the remainders to inter-reduce them and find new basis elements with new leading terms.

We believe this approach will scale and parallelize better than existing methods because the matrices in F4/F5 are extremely large and mostly triangular. The columns which have a triangular structure correspond to the reducible monomials, and step-by-step elimination will have very low arithmetic intensity, which is hard to parallelize. The heap based division process effectively combines all of the row operations for an entire column into a single step using storage proportional to the *number* of rows. It has high arithmetic intensity and very low storage costs, needing only $O(\#q + \#r)$ storage where the number of quotient terms/reducible monomials/matrix rows is $\#q$, and there are $\#r$ remainder terms corresponding to the irreducible monomials. This cost is paid for each S-polynomial separately, so the overall storage can be tightly controlled.

What we give up, in this formulation, is the flexibility to subtract two partially reduced polynomials and cancel lower order terms. We believe that opportunity is rare. The gain is that we avoid the construction of extremely large sparse matrices and all of the problems associated with them.

This leaves the problem of what to do with the remainders. The picture we should have in mind is a large sparse rectangular matrix, with many columns and few rows. Our goal is to compute its reduced row echelon form. The standard method is to invert a leading block and multiply through. That is, working mod p, identify and extract the submatrix of pivots and compute its inverse using p-adic lifting. Multiplying by this inverse produces the result. This could be a basic building block for sparse linear algebra computations.

We propose to develop this new approach to Gröbner basis computation, with an eye towards improving the sparse linear algebra steps which can not be completely eliminated.

This project is suitable for a PhD student and/or PDF interested in computing Gröbner bases and high-performance computing. In the first year the student would implement, in C, division by a set of polynomials. We propose to modify our heap based fraction-free division algorithm from [24] to accomplish this. This would also give Maple division by a triangular set which would improve the Hensel lifting in [29] for polynomial factorization over number and function fields. In the second year we would look at the sparse linear algebra and the Gröbner basis algorithm. In the third year we anticipate having to redesign the algorithm and try again. Maplesoft personnel would provide help with the sparse linear algebra and parallel infrastructure support.

4: Multivariate polynomial data structure sub-project.

Table 1 shows that the polynomial data structures used by Magma, Maple, and Mathematica limit their performance on modern computers. With our new POLY data structure we see an opportunity for a quantitative leap in overall performance. This sub-project would first integrate POLY into the Maple kernel as the default. One goal is to improve parallel speedup of Maple library routines like polynomial factorization by reducing sequential overhead by improving the efficiency of Maple kernel routines.

Secondly, we propose to design high-performance routines for POLY. For example, we have identified the operation of extracting the coefficient of $(x-a)^k$ from a polynomial in x for a given integer a as key in the multivariate Hensel lifting algorithm which is used for polynomial factorization. Also, there are many operations in the Maple kernel for polynomials that could be parallelized in principle. For example, sorting, coefficient extraction, and multi-point evaluation. Some of these, e.g., multi-point evalutation, are suitable for a GPU. For sorting we currently use American flag sort, a version of radix sort, but this still takes up most of the time for several operations. We would like to try a parallel radix sort.

Thirdly, we propose also to extend the POLY data structure so that we can encode not only monomials like $x^2y^3z^4$ in a single machine word, but also terms like $x^3e^xy'(x)$, that is, to support functions, as well as variables, e.g., $\sin(x)$, $f_x(x,y)$, and $RootOf(Z^2-2)$. This means that we can represent differential equations compactly and manipulate them more efficiently too. This would potentially benefit Maplesoft's other main product `MapleSim` which is used for industrial modelling and simulation. By inclusion of $RootOf(Z^2-2)$, which is Maple's representation for the algebraic number $\sqrt{2}$, we propose to include support for algebraic numbers. Currently Maple has no kernel support for computing with algebraic numbers. These extensions would involve a significant amount of coding but would potentially yield a ten-to-hundred-fold speedup for more general symbolic objects.

This project is suitable for a Masters student and/or PhD student interested in the design of symbolic computation systems and high-performance computing. In the first year the student(s) would focus on high-performance routines then in the second year supporting either derivatives or algebraic numbers. In the third year the student(s) and we will need to integrate the software into the Maple library. Maplesoft will participate in the design and integration of the new data structure, and parallel infrastructure support needed.

Team Expertise

Explain how the knowledge and experience of each team member relates to the expertise needed to accomplish the project objectives, and how the contributions of the team members (and company personnel) will be integrated.

The research area of each team member below includes computer algebra. Dr. Michael Monagan has expertise in all areas of this proposal. He has worked on the Maple and Magma computer algebra systems. Dr. Petr Lisonek has expertise in algebraic combinatorics. He will contribute to sub-project 2. Dr. Jürgen Gerhard is head of the Research Group at Maplesoft. He is co-author of the text *Modern Computer Algebra* [15]. He will contribute to the design of all four sub-projects. Dr. Allan Wittkopf of Maplesoft has expertise in solving differential equations. He is the author of Maple's library for linear algebra over finite fields. He would participate in the design of sub-projects 2 and 3. Mr. Darin Ohashi of Maplesoft will help with the design and implementation of the parallel support infrastructure needed for all four sub-projects. Ms. Clare So of Maplesoft will oversee the code (Maple and C) integration and testing.

Training of Highly Qualified Personnel

Describe how the knowledge and experience gained by graduate students, postdoctoral fellows, research assistants or others, including company personnel, is relevant to the advancement of the field, to developing practical applications of knowledge, or strengthening the industrial research base.

There are four good research and development problems in the scope of this proposal. For students, company personnel, and other research personnel, this project will provide excellent training in computational algebra with a balance between algorithm development and practical implementation. It involves writing software for polynomial computations over different rings from scratch. Few students will have such an opportunity. The work on parallel algorithm design and implementation will advance the field of computer algebra and benefit industry.

Each of the four sub-projects is suitable for one PhD student. The total number of students we anticipate supporting is 1 PDF, 4 PhDs, 1 Masters and 1 undergraduate student. Students we have in mind include: Mr. Lucas Jiaxiong, a current PhD student, has started working on sub-project 1. Mr. Baris Tuncer, a new PhD student, will work on sub-project 2 or 4. Mr. Roman Pearce, a possible PhD student, who has worked on the POLY data structure, could work on subproject 3 or 4. Mr. Rafid Abdullah, a possible PhD student, could work on sub-projects 2 or 4. Dr. Mahdi Javadi (PhD 2011), a possible PDF, could work on sub-projects 1, 2 and 3. He would help lead the parallel algorithm development.

Dr. Monagan will be teaching our Computer Algebra course in January 2013 to senior undergraduate students and graduate students and Dr. Lisonek plans to teach our Cryptography course in the Fall of 2013, also to senior undergraduate and graduate students. We expect to attract at least one undergraduate student and possibly one graduate student to the project through these courses.

Value of the Results and Industrial Relevance

Describe the anticipated value of the project results, highlighting the relevance of the scientific or technical advances, or the innovative techniques, processes or products that will be developed. Show how the outcome will address a current or future industrial or market need. Describe how the exploitation of the project results will benefit the Canadian economy within a reasonable time.

The main expected results of this project are

1. Fast modular algorithms for sparse polynomial GCD computation and with parallel support for multi-core platforms. This is necessary to enable computation with larger polynomials in Maple than is currently possible.

2. A new C library for polynomial arithmetic in $\mathbb{Z}_p[x]$ which has asymptotically fast algorithms and supports multi-core platforms. This is needed for 1 above but will also improve performance of root finding and factorization over finite fields.

3. A new C library for structured sparse linear system solving with application to fast Gröbner basis computation via the F4/F5 algorithms. A new heap based division code for division by a set of polynomials (possibly with parameters) with application to factorization over number and function fields and Maple's `Groebner` and `RegularChains` packages for solving systems of polynomial equations.

4. Incorporation of new polynomial data structure as the default will provide a big boost to Maple's overall efficiency thereby giving Maple a clear advantage over other computer algebra systems. Extending the new polynomial data structure to support functions will increase the range of applications in Maple which will benefit. Supporting derivatives will improve `MapleSim`'s efficiency for industrial modelling and simulation.

Maple has very limited parallel support right now. This project would add another significant amount of parallelism to Maple, which is very important because the sequential speed of computers will not increase much in the future, but the number of cores will.

Each of these four sub-projects will improve the performance of one or more of Maple's core algebraic facilities which will contribute to improving Maple's competitiveness. This will bring immediate and sustained benefit to Canadian industry because Maple is Maplesoft's main product. It will also make Maple more attractive to the research community as a vehicle for research and development.

Benefit to Canada

Outline any additional economic, social, and/or environmental benefits that will or could be realized in Canada.

The project will help train Canadian HQP with skills in parallel and high performance computing, skills in high demand in Canadian industry. We believe that the training in computational mathematics will also be of value to Canadian industry. Another indirect benefit is that by improving Maple (and also MapleSim) we will help improve STEM education in Canada as Maple is used in many mathematics, science and engineering departments in colleges and universities across Canada for training students in mathematics.

References

[1] W. S. Brown, On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors, *J. ACM* **18** (1971), pp. 476–504.

[2] R. Zippel, Probabilistic algorithms for sparse polynomials, *Proc. EUROSAM '79*, Springer-Verlag LNCS, **2** (1979), pp. 216–226.

[3] P. Wang. The EEZ-GCD Algorithm. *SIGSAM Bulletin*, **14** (2) pp. 50–60, 1980.

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

[5] J. de Kleine, M. B. Monagan, A. Wittkopf. Algorithms for the Non-monic case of the Sparse Modular GCD Algorithm. *Proc. ISSAC '05*, ACM Press, pp. 124–131, 2005.

[6] Mohammed Rayes, Paul Wang and Kenneth Weber. Parallelization of the sparse modular GCD algorithm for multivariate polynomials on shared memory multiprocessors. *Proc. ISSAC '94*, ACM Press, pp. 66–73, 1994.

[7] M. J. Encarnacion, Computing GCDs of Polynomials over Algebraic Number Fields, *J. Symbolic Computation* **20** (1995), pp. 299–313.

[8] P. Wang, M. J. T. Guy, J. H. Davenport, *p-adic Reconstruction of Rational Numbers*, in SIGSAM Bulletin, **16**, No 2 (1982).

[9] M. van Hoeij, M. B. Monagan. Algorithms for Polynomial GCD Computation over Algebraic Function Fields. *Proc. ISSAC '2004*, ACM Press, pp. 297–304, 2004.

[10] Mahdi Javadi and Michael Monagan. A Sparse Modular GCD Algorithm for Polynomial GCD Computation over Algebraic Function Fields. Appeared in *Proc. ISSAC '07*, ACM Press, pp. 187–194, 2007.

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

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

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

[14] Mahdi Javadi and Michael Monagan. Parallel Sparse Polynomial Interpolation over Finite Fields. *Proc. PASCO '2010*, ACM Press, pp. 160–168, 2010.

[15] J. von zur Gathen and J. Gerhard, *Modern Computer Algebra*. U. of Cambridge Press, 1999.

[16] M. Monagan. In-place arithmetic for polynomials over $\mathbf{Z}_n$. *Proc. DISCO '92*, Springer-Verlag LNCS, **721**, pp. 22–34, 1993.

[17] Victor Shoup. NTL: A Library for doing Number Theory. `www.shoup.net/ntl/`

[18] David Harvey. zn_poly, a C library for polynomial arithmetic in $\mathbb{Z}/n\mathbb{Z}[x]$. `http://web.maths.unsw.edu.au/~davidharvey/code/zn_poly`

[19] William Hart. FLINT: Fast Library for Number Theory. `www.flintlib.org/`

[20] 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.

[21] 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.

[22] M. Moreno Maza and Y. Xie. FFT-based Dense Polynomial Arithmetic on Multicores. Proc. HPCS 2009, Springer-Verlag LNCS **5976**, 378-399, 2010.

[23] M. Moreno Maza and W. Pan. *Fast polynomial arithmetic on a GPU*, volume 256. J. of Physics: Conference Series, 2010.

[24] Michael Monagan and Roman Pearce. Sparse Polynomial Division Using a Heap. *J. Symb. Cmpt.* **46** (7) 807–822, 2011.

[25] Michael Monagan and Roman Pearce. Parallel Sparse Polynomial Multiplication using Heaps. *Proc. ISSAC '09*, pp. 263-269, ACM Press, 2009.

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

[27] Michael Monagan and Roman Pearce. Sparse Polynomial Multiplication and Division in Maple 14. *Communications in Computer Algebra*, **44**:4, 205–209, December 2010.

[28] Michael Monagan and Roman Pearce. POLY: A new polynomial data structure for Maple 17. ISSAC 2012 Software Session. To appear in *Communications of Computer Algebra* **46**(4) 2012.

Preprint: `http://www.cecm.sfu.ca/~mmonagan/papers/issac12.pdf`

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

[30] B. Buchberger, *Groebner bases: an algorithmic method in polynomial ideal theory.* Multidimensional Systems Theory, edited by N. K. Bose, D. Reidel Publishing Company, Dordrecht, pp. 184-232, 1985.

[31] A. Giovini, T. Mora, G. Niesi, L. Robbiano, and C. Traverso, "One sugar cube, please" OR Selection strategies in the Buchberger algorithm. *Proc. ISSAC '91*, ACM Press, pp. 41–29, 1991.

[32] Jean-Charles Faugère, A New Efficient Algorithm for Computing Gröbner Bases (*F4*). *J. Pure and Applied Algebra*, **139**, pp. 61–83, 1999.

[33] Thomas Yan. The geobucket data structure for polynomials. *J. Symb. Cmpt.* **25**)(3): 285–293, 1998.

[34] S.C. Johnson. Sparse polynomial arithmetic. *ACM SIGSAM Bulletin*, **8** (3) 63–71, 1974.

[35] B. Hammersholt Roune and M. Stillman. Practical Groebner Basis Computation. To appear in *Proc. ISSAC 2012*, 2012.

[36] Jean-Charles Faugère, A New Efficient Algorithm for Computing Gröbner Bases Without Reduction to Zero (*F5*). *Proc. ISSAC '02*, ACM Press, pp. 75–83, 2002.

[37] Jean-Charles Faugère. FGb: a library for computing Gröbner bases. Mathematical Software ICMS 2010, Springer-Verlag LNCS **6327** pages 84-87, September 2010.

[38] Jean-Charles Faugère and Sylvain Lacharte. Parallel Gaussian Elimination for Gröbner bases computations in finite fields. *Proc. PASCO '10*, ACM Press, 89–97, 2010.

[39] S. Pohlig and M. Hellman. An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance. *IEEE Trans. on Information Theory*, 24:106−110, 1978.

[40] Paul Wang. An Improved Multivariate Polynomial Factoring Algorithm. *Math. Comp.* **32**(44) 1215–1231, 1978.

[41] Mark Giesbrecht, George Labahn, and Wen-shin Lee. Symbolic-numeric sparse interpolation of multivariate polynomials. *J. Symb. Cmpt.*, **44**, 943–959, 2009.

[42] Granlund, T., 2008. The GNU Multiple Precision Arithmetic Library, version 4.2.2. `http://www.gmplib.org/`

[43] Mahdi Javadi and Michael Monagan. Parallel Sparse Polynomial Interpolation over Finite Fields. Submitted to *J. Symb. Comp.*, 2010.
Preprint: `http://www.cecm.sfu.ca/~mmonagan/papers/ParallelInt.pdf`

[44] J. Vershelde and G. Yoffe. Evaluating polynomials in several variables and their derivatives on a GPU computing processor. Submitted to arXiv:1201.0499v1, January 2012.

[45] M. van Hoeij, M. B. Monagan, A Modular GCD Algorithm over Number Fields Presented with Multiple Field Extensions. *Proc. ISSAC '2002*, ACM Press, pp. 109–116, 2002.