Without the support of NSERC for the past five years, my research program would not have been so successful. It is a pleasure to be able to conduct my scientific investigation in Canada where there is an established tradition of excellent, funded research in computer algebra. My research is devoted to the design and implementation of polynomial system solvers based on symbolic computation. Solving systems of non-linear, algebraic or differential equations, is a fundamental problem in mathematical sciences. It has been studied for centuries and still stimulates many research developments.

Solving polynomial systems is a driving subject for symbolic computation. In many computer algebra systems, the `solve` command involves nearly all libraries in the system, challenging the most advanced operations on matrices, polynomials, algebraic numbers, polynomial ideals, etc.

Symbolic solvers are also powerful tools in scientific computing: they are well suited for problems where the desired output must be exact and they have been applied successfully in mathematics, physics, engineering, chemistry and education, with important outcomes. See Chapter 3 in [10] for an overview of these applications. While the existing computer algebra systems have met with some practical success, symbolic computation is still under-utilized in areas like mathematical modeling and computer simulation. Part of this is due to the fact that much larger and more complex computers are required - often beyond the scope of existing systems.

The implementation of symbolic solvers is, indeed, a highly difficult task. Symbolic solvers are extremely time-consuming when applied to large examples. Even worse, intermediate expressions can grow to enormous size and may halt the computations, even if the result is of moderate size. Therefore, the implementation of symbolic solvers requires techniques that go far beyond the manipulation of algebraic or differential equations; these include efficient memory management, data compression, parallel and distributed computing, etc.

Though there has been progress, researchers have realized that solvers, and other computer algebra software, were far from making the best use of their computer resources. Indeed, computer algebra software packages rarely take into account memory hierarchy and memory contention. The increasing diversity of architectures makes this problem more dramatic. With the desire of achieving high-performance on practical problems, research in symbolic computation has entered a new era where implementation techniques are at least as important as theoretical developments. This phenomenon is largely accentuated by the recent improvements in computer hardware which focus on parallel processing rather than increasing clock speed alone.

The goal of my proposed research is to deliver efficient mathematical algorithms, implementation and code optimization techniques for symbolic polynomial system solvers, and more generally for symbolic computation, to achieve high-performance on modern computing resources from SMPs and multi-cores to clusters of multi-processors. To achieve this, I propose the following projects, which extend my previous work into five promising and strongly-related areas.

FASTTRIADE**:** Fast arithmetic and modular methods for triangular decomposition,

COGEPAS**:** Code generation for polynomial arithmetic subroutines,

PASCOLIB**:** Parallel symbolic computation library,

HPCSOLVE**:** High-performance solver for clusters of multi-processor,

CADYNA**:** Computer algebra support for studying dynamical systems.

In the sequel, references written in boldface as **[1]**, **[2]**, ... refer to the contribution list in my Form 100. Those numbered as [1], [2], ... are listed at the end of this proposal.

1 Overview

The development of polynomial system solvers, as computer programs based on symbolic computation, started four decades ago with the discovery of Gröbner bases in the PhD thesis of B. Bucherber, whereas efficient implementation capable of tackling *real-world applications* is very recent [6]. Triangular decompositions are an alternative way for solving systems of algebraic equations symbolically. They focus on extracting geometrical information from the solution set $V(F)$ of the input polynomial system F rather than insisting on revealing algebraic properties as Gröbner bases do. A triangular decomposition of $V(F)$ is given by finitely many polynomial sets, each of them with a triangular shape and so-called a triangular set[1]; these sets describe the different components of $V(F)$, such as points, curves, surfaces, etc. Triangular decompositions were invented by J.F. Ritt in the 30's for systems of differential polynomials. Their stride started in the late 80's with the method of W.T. Wu dedicated to algebraic systems. Different concepts and algorithms extended the work of Wu. At the end of 90's the notion of a regular chain, introduced by M. Kalkbrener, led to important algorithmic improvements. The era of polynomial system solvers based on triangular decompositions could commence. Since 2000, exciting complexity results [3] and algorithms (see the progress report below) have boosted the development of implementation techniques. I believe that triangular decompositions are highly promising techniques which have the potential to produce high-performance solvers.

A few sequential computer algebra packages (GMP, NTL, Gb/FGb) can be regarded as high-performance tools. However, only the first one, which provides big integer arithmetic, can support the proposed software. The largest effort of the computer algebra community toward high-performance is on parallel symbolic computation. This has been a very active area during the 80's and 90's, see the survey in Section 2.18 of [10]. Unfortunately, the attention of the computer algebra community has been somewhat withdrawn from parallelism during the last decade. The success of the workshop *Parallel Symbolic Computation 2007* (which I organized last Summer) indicates a renewed interest.

Two goals rule the design of a parallel software package: the application must exhibit enough parallelism to efficiently utilize multiple processors while it must minimize the data communication costs and memory traffic. A number of features of symbolic computation make this design even more complex. The latter three are typical difficulties in polynomial system solving.

Data duplication. In a sequential program, the data-structures representing two polynomial sets $S = \{p, q\}$ and $T = \{q, r\}$ are likely to share the representation of the common element q; data duplication may not be avoided in a parallel program running two independent processes, one holding S and the other T. Data-structures like the above sets, or linked lists, are frequent in symbolic computation and this problem of data duplication can be a bottleneck.

Huge data. Intermediate expressions can become huge in a nearly unpredictable way. Thus, data serialization and unserializatiom (for data communication) have an appreciable cost. More generally, memory traffic and network communication are harder to handle in this context.

Task irregularity. In an algorithm using a task pool (say the Bucherber Algorithm for Gröbner bases managing a list of critical pairs) the number of tasks and their grain sizes may vary dramatically and are also hard to determine in advance; this makes task scheduling quite difficult.

Evolving job. When running a parallel symbolic solver, the total amount of computing resources that can be efficiently utilized is far from being constant during the execution time. Addressing this issue is actually a challenge for dynamic job schedulers; see the survey papers [7, 15].

[1] This notion extends to non-linear systems that of a triangular system, well-known in linear algebra.

2 Progress report

The expected results identified in my previous Discovery grant have been achieved. First, we have obtained a coarse-grained parallel solver for triangular decompositions **[6]** together with a supporting framework library **[7]**. Data duplication, as defined above, has been recognized as a major issue: one of the objectives of our PASCOLIB project is to face this difficulty. Secondly, we have obtained the first modular method (allowing a good control on intermediate expression swell) for computing triangular decompositions **[24]**. This algorithm applies only to input systems with rational number coefficients and with finitely many solutions. The cases of systems with coefficients modulo a prime number or with infinitely many solutions are among the objectives of our FASTTRIADE project.

We have also obtained unanticipated results, which are major tools for reaching high-performance. First, we have identified implementation techniques for fast polynomial arithmetic operations **[8, 17]**: our code compares to, and often outperforms, the packages with similar specifications **[14]**. Secondly, we have discovered highly efficient algorithms for low-level operations supporting triangular decompositions, such as polynomial multiplication modulo a triangular set **[5]**. Developing auto-tuning software tools for adapting these low-level routines on the main stream and emerging architectures is the objective of our COGEPAS project. Another unanticipated result is a new algorithmic approach for studying polynomial systems with parameters **[11]**. This is essential to our CADYNA project.

3 Objectives

It is well recognized that the impact of the high-performance software for both numerical and exact linear algebra on engineering and scientific computing is tremendous. The most significant results are the BLAS [11], LAPACK [4], PLAPACK [1], ATLAS [19] and LinBox [5]. The successful techniques include hierarchical modularization, implementation of highly efficient algorithms for basic routines, block-based algorithms for better exploiting the memory hierarchies, parallel and distributed methods, as well as automatically optimizing and tuning code on different platforms.

Today, solving polynomial systems of non-linear equations on high-performance architectures is of great interest. On one hand, exciting progress has been made on practical aspects of symbolic solving. On the other hand, the pervasive ubiquity of parallel architectures has led to a new quest for mathematical algorithms and software capable of exploiting these computing resources. Therefore, it is time to realize a high-performance symbolic solver so that the scientific computing community could readily take advantage of its power and more difficult problems could be solved.

Inspired by the successful story of high-performance linear algebra, based on my experience with the development of three sequential solvers and one coarse-grained parallel solver, as well as my recent contributions to implementation techniques, fast algorithms, modular methods and fine/medium-grained parallelism, I propose five strongly-related projects addressing high-performance in symbolic computation. The core project HPCSOLVE will deliver polynomial system solvers for clusters of multi-processors to compute triangular decompositions. It will rely on three foundamental projects, FASTTRIADE, COGEPAS and PASCOLIB dedicated respectively to algebraic algorithms, code generation and parallel symbolic computation. Our fifth project, CADYNA, will produce an expert system for studying dynamical systems. It will rely intensively on HPCSOLVE and thus will be our driving application. Today, no software possesses the functionalities and computing power of the proposed one. I strongly believe that these five projects could bring major advances in symbolic computation and a great opportunity for computer algebra to conquer new application areas.

FASTTRIADE**: Fast arithmetic and modular methods for triangular decomposition.** The research commenced in **[5]**, on fast algorithms for low-level operations supporting triangular decompositions will be continued. Similarly, following **[24]**, the quest for modular methods has to be pursued. It is well-known that modular methods favor the use of fast arithmetic. As demonstrated in **[6]**, modular methods also creates opportunities for parallel execution. Therefore, modular methods are essential techniques for reaching high-performance in symbolic computation. As mentioned earlier, the fundamental situation of polynomial systems, with coefficients modulo a prime number and with finitely many solutions, still requires an efficient modular method for computing triangular decompositions. In the Master's Thesis of my PhD student Raqeeb Rasheed [17], we have obtained promising results for systems with 2 or 3 variables. I believe that our strategy generalizes to systems in n variables and n equations. First, we concentrate on polynomial systems F which satisfy what we call *genericity assumptions*. Under these, we obtain a highly efficient algorithmic solution, where the cost of each operation can be controlled in a nearly optimal way. Secondly, we relax these genericity assumptions while maintaining an efficient algorithm. In the work of G. Lecerf [12] genericity assumptions are also used; relaxing them is achieved by means of generic perturbation. We avoid this transformation by making use of our recent algorithms for *change of variable ordering* **[3]** and *triangular set merging* **[24]**. The experimental results of [17] illustrate the good behavior of our approach.

COGEPAS**: Code generation for polynomial arithmetic subroutines.** Similarly to the ATLAS [19] and SPIRAL projects [16], COGEPAS will be dedicated to code generation, code optimization and platform adaptation for a specific domain of routines. We will focus on arithmetic operations (multiplication, polynomial GCD, etc.) modulo a triangular set. These operations are easy to implement in a high-level language. Reaching high-performance, however, requires to take care of different issues and the proposed software tool will have an *algorithm level* and an *implementation level*, as in SPIRAL. By means of a syntax directed-approach, the algorithm level will generate different algorithmic solutions, for instance one using FFT-based arithmetic, and another using classical arithmetic. These solutions, encoded in a high-level language, will be compared, first, by a complexity analysis and secondly by benchmarking their generated C code. Then, thresholds will be determined, depending on the input data size, and an algorithmic solution will be selected. This algorithm level will automatize the work that we conducted manually in **[9]**. At the implementation level, according to the features of the target architecture, compiler optimization techniques such as code transformation for locality and parallelism are adjusted to our application. The generated C code is augmented with either CILK [8] or KAAPI [9] constructs (for task scheduling). Then, the code is profiled on the target multi-processor machine. If a bottleneck is identified, then additional phases of code optimization are performed.

PASCOLIB**: Parallel symbolic computation library.** The goal of the PASCOLIB project is to provide a high-performance software library for symbolic computation on clusters of multi-processors. This library, written in C and ALDOR, will extend the framework introduced in **[7]** and will provide both multi-threaded and multi-processed parallelisms. Moreover, COGEPAS will support the PASCOLIB in a similar way that ATLAS supports LAPACK. A typical PASCOLIB implementation would be that of a modular method for computing a polynomial GCD modulo a parametric triangular set (or regular chain). Such an algorithm could specialize the parameters to several values, leading to several threads that can run concurrently the routines of COGEPAS. The task scheduling could rely fully on CILK or KAAPI. However, certain algorithms, in particular those for triangular decomposi-

tions, can efficiently guide the task scheduling (based on information discovered during the computation, see **[6]**) and the PASCOLIB should accommodate them. Another feature of the PASCOLIB would be distributed data-structures in order to limit data duplication (as defined above). Suppose that, during a computation, a Master M and its Workers $W_1, W_2, \ldots, W_p$ need to exchange data of some recursive type T, say binary tree. Suppose that M decides to send a copy of a variable v of type T to W_1. Then, M should send to W_1 only the sub-expressions of v that W_1 does not know such that W_1 can construct a copy of v, without duplicating any subtrees of v in its local memory. If the type T is declared "distributed", then the PASCOLIB would provide this simple mechanism for objects of T in a transparent way. This mechanism should be very effective for types T like polynomial sets.

HPCSOLVE**: high-performance solver on clusters of multi-processors.** Based on my experience on coarse-grained parallel triangular decompositions and medium/fine-grained parallel arithmetic operations modulo a triangular set, I am convinced that triangular decompositions will lead to the realization of HPCSOLVE, the first symbolic solver with multi-level parallelism. Following the techniques initiated in **[6]**, the modular algorithms for polynomial system solving developed by the FASTTRIADE project will create opportunities for coarse-grained parallel execution. These modular algorithms, implemented in PASCOLIB, will rely on the low-level routines provided by COGEPAS. As shown in **[8]**, these routines exhibit a rich parallelism. The high productivity of HPCSOLVE will result not only from its multi-level parallelism, fast polynomial arithmetic operations and modular methods implemented in COGEPAS and PASCOLIB but also from the fact that COGEPAS code is optimized for locality and parallelism, and tuned on the target machine. Despite of all its high-performance features, HPCSOLVE should have irregular computing resource needs during an execution. Developing HPCSOLVE together with a job scheduler accommodating evolving jobs (as defined above) would be of great interest.

CADYNA**: Computer algebra support for studying dynamical systems.** The existing software packages devoted to dynamical systems, based on symbolic or numerical methods, fall in the following categories: (a) computation of normal forms and center manifolds using [13], (b) stability analysis of equilibria using symbolic techniques [18], (c) stability analysis of equilibria and limit cycles for planar dynamical systems [2, 20], (d) symbolic computation of Lyapounov quantities [14], (e) computation of bifurcation diagrams using numerical parameter continuation (Content, Matcont, AUTO 2000, XPP/XPPAUT). All these packages perform only specific tasks and frequently assume that particular conditions hold. In addition, they do not have mechanisms to automatically produce systematic case distinctions according to parameter values. The recent progress for solving parametric polynomial systems indicate that a computer program could efficiently generate an automatic and systematic discussion of the locus of the equilibria, together with the normal forms at these equilibria, of a given (polynomial) dynamical system, depending on the values of its bifurcation parameters. The objective of the CADYNA project is to design and implement the first expert software system of this kind.

Summary and impact. We will design and deliver high-performance software packages: solvers for polynomial systems and libraries for symbolic computation. The problems arising from our driven application will put all the components of our solvers and libraries at extreme challenges, bringing efficiency and robustness. I believe that the technology and software generated by this project will be major advances in computer algebra. They will also provide scientific computing with tools capable of tackling problems for which no software solutions exist today.

References

[1] P. Alpatov, G. Baker, C. Edwards, J. Gunnels, G. Morrow, J. Overfelt, R. van de Geijn, and Y. J. Wu. Plapack: parallel linear algebra package design overview. In *Proc. of the 1997 ACM/IEEE Conference on Supercomputing*, pages 1–16. ISBN:0-89791-985-8, ACM Press, NY USA, 1997.

[2] A. Back, J. Guckenheimer, M. Myers, F. Wicklin, and P. Worfolk. dstool: Computer assisted exploration of dynamical systems. *Notices Amer. Math. Soc.*, 39:303–309, 1992.

[3] X. Dahan and É. Schost. Sharp estimates for triangular sets. In *ISSAC 04*, ACM, 2004.

[4] J. Dongarra and J. Wasniewski. Lapack95 - high performance linear algebra package. *Mathematical Modeling and Analysis*, 5(2000):44–54, 2000.

[5] J.-G. Dumas, T. Gautier, and C. Pernet. FFPACK: Finite field linear algebra subroutines. In *ISSAC 04*, ACM, 2004.

[6] J.-C. Faugère. *Résolution des systèmes d'équations algébriques*. PhD thesis, Univ. Paris 6, 1994.

[7] D. G. Feitelson. A survey of scheduling in multiprogrammed parallel systems. Technical Report Report RC 19790 (87657), IBM T. J. Watson Research Center, 1994. Revised August 1997.

[8] M. Frigo. Multithreaded programming in cilk. In *PASCO'07*, ACM Press, 2007.

[9] T. Gautier. Kaapi: a thread scheduling runtime system for data flow computations on cluster of multi-processors. In *PASCO'07*, ACM Press, 2007.

[10] J. Grabmeier, E. Kaltofen, and V. Weispfenning. *Computer Algebra Handbook*. Springer, 2003.

[11] C. L. Lawson, R. J. Hanson, D. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for fortran usage. *ACM Trans. Math. Soft.*, 5(1979):308–323, 1979.

[12] G. Lecerf. Computing the equidimensional decomposition of an algebraic closed set by means of lifting fibers. *J. Complexity*, 19(4):564–596, 2003.

[13] X. Liu. *Symbolic Tools for Analyzing of Nonlinear Dynamical Systems*. PhD thesis, UWO, 1999.

[14] S. Lynch. Symbolic computation of Lyapunov quantities and the second part of Hilbert's sixteenth problem. In *Differential Equations with Symbolic Computation*, Trends in Mathematics, pages 1–22, Basel, Boston, Berlin, 2005. Birkhäuser Verlag.

[15] S. Majumdar and E. W. Parsons. *Performance Evaluation: Origins and Directions*, volume LNCS 1769, in *Pallel Job Scheduling: A Performance Perspective.* Springer, 2004.

[16] M. Püschel, J. M. F. Moura, J. Johnson, D. Padua, M. Veloso, B. W. Singer, J. Xiong, F. Franchetti, A. Gačić, Y. Voronenko, K. Chen, R. Johnson, and N. Rizzolo. SPIRAL: Code generation for DSP transforms. *Proceedings of the IEEE*, 93(2):232–275, 2005.

[17] R. Rasheed. Modular methods for solving polynomial systems, 2007. U. of Western Ontario.

[18] D. Wang and B. Xia. Stability analysis of biological systems with real solution classfication. In *ISSAC'2005*, ACM Press, 2005.

[19] R. C. Whaley, A. Petitet, and J. Dongarra. Automated empirical optimization of software and the atlas project. *Parallel Computing*, 27(1-2):3–35, 2001.

[20] P. Yu. *Maple Source Codes and Input Files for Computing Normal Forms*. http://pyu1.apmaths.uwo.ca/∼pyu/pub/index/software.html, 2004.