

Computational Algebra for Commodity Parallel Machines (The ParaGAP Project) Further Particulars

Kevin Hammond and Steve Linton
University of St Andrews

22 July 2002

1 Overview

Abstract algebra is one of the most powerful unifying paradigms of twentieth century mathematics. A vast field of research in its own right, it is also a key tool in other areas of mathematics, in physics, in chemistry and in computer science. In the last thirty years, computational tools have played an increasing role in the development and application of abstract algebra, until today, computational algebra systems are in widespread use across a range of theoretical and applied disciplines.

GAP (Groups, Algorithms and Programming) [34] is a leading computational algebra system, and the leading open source system. It supports computation in many areas of algebra with particular emphasis on group theory. GAP has a well established international user base, and is used for teaching, research and commercial applications including cryptography, crystallography, the study of quantum computing, and applications in many areas of mathematics. Many GAP users need to perform very large computations, whether in theoretical development, such as exploring very large groups, or in applications, such as managing the symmetry in very large searches. This creates a requirement for GAP to support a range of low cost high performance computing solutions, which, today, inevitably means multi-processor systems of one form or another. On the other hand, since most GAP users are not programming specialists, let alone parallel programming specialists, it is essential that details of the parallel implementation: distribution of the computation, message-passing, synchronization and so on be hidden from the user.

The ParaGAP project is a 3-year project to link front-end GAP applications with a high-level, portable and well-developed parallel back-end, the GUM implementation [37] of Glasgow Parallel Haskell (GpH) [37]. This will support high-performance computation via the established and popular GAP user interface on economical platforms such as multi-processor workstations and Beowulf-style PC or workstation clusters. Such hardware is becoming increasingly available to the academic and general research community that forms the primary GAP user base. The two systems will be linked through a flexible general architecture offering the GAP programmer facilities for transferring appropriate types of data to and from GpH and for initiating general parallel computations in GpH. The interface will exploit the OpenMath standard, as appropriate.

In order to validate this architecture, and to deliver benefits to end users, the project will develop and implement parallel algorithms for a number of “key sub-problems” which form

core time-consuming steps in a very wide range of computations. Efficient algorithms for these will consequently be of very general benefit, even to GAP users who do not wish to do any new programming at all. The problems (see Section 6.1) have been selected to reflect a variety of parallel computations, varying, for example, in the regularity of the computational and/or data structures, in the balance of memory and processor requirements, and in load distribution patterns.

In addition to the direct research results obtained through the parallelisation of these key sub-problems, we also anticipate that the project will deliver enhanced insights into the underlying sequential algorithms, that it will provide a valuable source of performance information for this complex and rarely studied class of parallel algorithms, and that the key sub-problems will serve as good motivating applications for researchers in functional programming.

The ParaGAP project represents an equal inter-disciplinary collaboration between pure mathematicians, under the leadership of Steve Linton, a main developer of GAP, and computer scientists, under the leadership of Kevin Hammond, a main developer of GpH.

2 Aims and Objectives

2.1 Aims

The aims of the ParaGAP project are:

- To explore the parallelisation of typical computations arising in computational abstract algebra;
- To provide usable support for parallelism in the popular, flexible, extensible and interactive GAP problem-solving environment.

2.2 Objectives

The objectives of the project are:

- To interface the computational algebra system GAP to GpH, an instrumented implementation of a parallel functional language;
- To develop high-level programming constructs allowing the direct exploitation of parallelism from within GAP algorithms;

- To develop and implement parallel algorithms for a range of key problems in computational algebra (see Section 6.1).

algebra systems (such as Maple [7] and Axiom [17]) have a limited abstract algebra capability, but they are quite inadequate for research in this area.

3 The Research Fellow

This is an excellent opportunity for a recent doctorate to extend their research experience working on an exciting and innovative project. The successful Research Fellow will be appointed to work on the project for up to 3 years starting in Autumn 2002. The ideal candidate would possess a doctoral degree in a relevant research area, with demonstrable programming skills both in functional programming and in C. Knowledge of the GHC Haskell compiler and runtime system would be an advantage, but appropriate allowance will be made for an otherwise well-qualified candidate. The post involves constructing group theoretic applications, and thus some university-level experience in discrete mathematics and a willingness to learn new concepts is essential. Knowledge of group theory would be an advantage, but is not required.

4 Existing Work

4.1 Computational Abstract Algebra

The use of computers in pure mathematics can be traced back to the 1950s (e.g. [10]). Early specialised programs then gave rise to a first generation of more flexible and general systems more suitable for use by the non-specialist (e.g. [5, 19, 28]), which have, in turn, been largely supplanted by more modern systems.

This development has paralleled that of so-called “computer algebra” systems (such as Maple [7] or Mathematica [39]) which focus on the formal manipulation of mathematical expressions and on analytic problems such as integration and solving differential equations. In contrast *computational abstract algebra systems focus on computation with permutations, finite field elements, words in abstract generators and other specialized data, and on the application of this to answer structural questions about groups, rings and algebras*. While there is some common ground in systems design and in areas such as polynomial rings, computer algebra systems and computational algebra systems have developed largely independently, and focus on different types of problems.

Computational algebra has played an important role in notable mathematical developments, for example in the Classification of Finite Simple Groups, formerly a major open research problem in mathematics whose solution led to the award of a number of Fields’ medals (similar in stature to Nobel prizes). Related computational methods are also essential in areas of mathematics which apply to computer science, such as formal languages, coding theory, cryptography and combinatorics.

Currently there are two general-purpose systems in widespread use: the GAP system which forms the basis of this proposal and the MAGMA system from Sydney [2]. The MAGMA system provides somewhat more wide-ranging features than GAP, but offers less support for application development, and is not open source. Some general computer

4.2 GAP

The GAP system [34], originally for computational group theory, but now supporting computational algebra and discrete mathematics, was developed from 1984 onwards at Lehrstuhl D für Mathematik, RWTH-Aachen, Germany, under the leadership of Prof. J. Neubüser. Since 1997, development has been coordinated in St Andrews, currently by the Centre for Interdisciplinary Research in Computational Algebra. Available for UNIX, MacOS and Windows, GAP has an excellent international reputation for quality, usability, reliability and flexibility [35]. The system is estimated to be in use at approximately 500–1000 institutions world-wide, including 50–80 in the UK, with users drawn from across the mathematical sciences and beyond. The system is free and open, with kernel and library sources available electronically by anonymous FTP or on CD.

4.3 Parallel Computer Algebra Systems

In the wider field of general purpose computer algebra, a significant body of research exists concerning specific parallel algorithms in some sub-areas, notably for term re-writing and Gröbner basis completion (see, for example [4] and [1]). A number of one-off parallel programs have been also developed for specific algebraic computations, mainly in representation theory [27].

There is, however, little if any support for parallelism in the most widely used computer algebra systems such as Maple, Axiom or indeed GAP. As research systems, Maple/Linda – Sugarbush [8] supports sparse modular gcd and parallel bignum systems, with Maple/DSC [6] supporting sparse linear algebra. The ParGAP package [11] offers support for very coarse-grained parallel computation using multiple, communicating GAP processes. None is in widespread use at present, however, and none supports the broad range of computational algebra applications we propose to target.

Among the less widely used systems, PARSAC-2 [18] is a computer algebra library implementing parallel Gröbner bases, PAC++/Givaro [12] is a C++ library aimed at linear algebra and algebraic number computation, which interfaces to PVM/MPI using the Athapascan parallel runtime environment, while Π^{it} [25] is a fine-grained, multithreaded library for the Aldor language, which can be called from within the Axiom computer algebra system [17]. None of these systems provides a fully general parallel implementation of finite fields, groups and permutations of the kind that we are proposing in this project. The tutorial by Roch and Villard [33] provides a good general survey of work in the field as of 1997.

4.4 GpH

GpH [37] is well-established as a leading parallel functional language implementation. It is built on the state-of-the-art GHC implementation [31] of the standard non-strict functional language, Haskell [29, 30]. GHC is a well-engineered research system that is available on a variety of platforms

including Windows, Macintosh and Unix platforms. GHC is widely used in the research community, being in use at several hundred institutions world-wide and is starting to be used commercially by e.g. Galois Connections (Oregon, USA). Haskell and GHC have recently received support from Microsoft as part of its .NET strategy.

GpH represents the latest in a line of parallel implementations developed from 1989 onwards at Glasgow under Prof. Simon Peyton Jones, and subsequently at St Andrews and Heriot-Watt under Kevin Hammond and Phil Trinder. It is now robust, possesses a good range of high-level performance monitoring tools, provides simulation support for prototyping, tuning, or debugging [15], supports very high-level programmable models of parallelism [36], has been ported to shared-memory machines, distributed-memory machines and clusters of workstations [38], and delivers good speedups over its sequential parent. Importantly for the purposes of this project, it provides a good environment for prototyping and developing new parallel algorithms, having been tested on a variety of realistic applications [23].

5 Novelty and Timeliness

The primary novelty of this work lies in the fact that, for the first time, a group theoretic algebra package will support a straightforward, efficient and flexible interface to parallelism. This potential has not so far been realised—this project intends to remedy this situation.

In addition, we will be developing new parallel algorithms for a range of important calculations in computational algebra. In many cases these will be the first parallel algorithms created to solve these problems; in other cases, they will be the first flexible and general-purpose parallel algorithms.

Finally, the work is novel in representing a new, but particularly well-matched application of functional programming language technology: the work offers the prospect of developing completely novel parallel algorithms using the non-strict functional language, Haskell, and of delivering real benefit to a large number of users through improved performance and access to modern high-performance computer resources. Such benefit would almost certainly be much more costly to deliver using conventional approaches. The work is especially significant to the parallel functional programming community in offering, for the first time, access to a large user base with sophisticated problems in the field of symbolic computation.

The work is timely because of the current trend towards low cost commodity parallel hardware, which is affordable by the GAP user base. Designs based on shared memory or clusters of workstations are becoming pervasive at all levels of the computer market from PCs and workstations to high-end servers. Researchers in group theory are significantly limited in the computations they are able to perform by both memory requirements and execution time. This project will thus extend the computational boundaries of computational algebra software, enabling new research to be carried out.

While not formally comprising part of the E-Science initiative, the project clearly fits into its spirit. The project aims to enhance the capabilities of mathematicians and other scientists through the exploitation of advanced parallel computer software for computational algebra. We also anticipate that later versions of the software developed in this project will be

capable of making rapid and effective use of emerging “Grid” technology.

6 Programme and Methodology

We propose to exploit the capabilities of the GpH parallel language and system in conjunction with the GAP computational algebra system in order to systematically explore the parallelisation of a number of key problems in computational algebra. GpH functions and libraries will be called directly from within GAP programs in order to provide support for parallel computations on shared-memory and distributed memory hardware. We will also provide programming language constructs allowing direct exploitation of parallelism from within GAP programs, allowing GAP programmers to develop additional parallel algorithms and applications. We will use a systematic prototyping approach to parallelisation, based on high-quality simulation/emulation plus real machine measurements. The performance of the algorithms will be carefully measured at each stage of development using existing high-quality instrumentation and visualisation techniques.

6.1 Programme of Work and Milestones

The proposed work programme is structured into three interconnected work packages (WP1–WP3), each with associated milestones and deliverables.

WP1: System Integration and Library Construction

In this workpackage, we will integrate the GAP system with GpH. This will allow GAP programs to call libraries and functions written in parallel Haskell, and will provide a callback facility to GAP routines, as appropriate.

We will develop Haskell libraries for finite fields, groups and permutations, that will be used to support parallel programs that can be called by GAP, which is not itself multi-threaded. In accordance with good software engineering practice, we will first develop a prototype interface and libraries and then refine these in the light of experience so as to yield the final versions. In order to enhance portability, provide scope for interaction with other systems such as Maple or Mathematica, and reduce design costs, we will use the standard OpenMath encoding for our prototype interface. This may subsequently be tuned for performance.

The milestones for this workpackage are: **M1.1** – prototype interface from GAP to GpH; **M1.2** – prototype Haskell libraries for basic computational algebra operations; **M1.3** – completed interface from GAP to GpH; and **M1.4** – completed Haskell libraries for basic computational algebra.

The corresponding deliverables are: **D1.1** – a system exploiting parallelism on commodity architectures that can be called from within GAP programs; and **D1.2** – implementations of Haskell libraries for finite fields, permutations and groups.

WP2: Parallel Programming Extensions for GAP

In this workpackage, the GAP language will be extended with parallelism constructs that can be directly exploited from within GAP programs. The constructs will be implemented as

key problem	parallel structure	data structures	libraries used
i)	irregular	large, mod. complex	P
ii)	regular	huge or complex	F
iii)	irregular	large, simple	W
iv)	irregular	large, complex	W

Table 1: Classification of key computational algebra problems – P = permutations library, W = word library, F = finite field library

GpH primitives. Our experience of designing and implementing very high level parallelism [36, 14] will guide the design and implementation of these primitives. The milestone for this workpackage (**M2.1**) is the development of parallelism constructs for the GAP system, and their implementation as GpH primitives. The associated deliverable (**D2.1**) is a working GAP system with direct support for parallelism.

WP3: Algorithm Development and Performance Measurement

We have identified a set of key sub-problems as meeting the following criteria: they are the main time-consuming steps of important end-user computations; their general structures are typical of those encountered in other important computations; and they may be amenable to parallel computation. As potential parallel algorithms, they display a variety of characteristics ranging from simple, regular parallel structures to complex, irregular structures with dynamic load distribution properties. These properties are summarised in Table 1. The problems we have identified as most suitable for initial study are:

- i) Computation of a base and strong generating set for a permutation group. This is a critical first step for almost any computation with a permutation group, and often the time-critical step.
- ii) Determining the orbit of a subspace of a vector space under the action of given matrices. This is an important sub-problem in representation theory and one which requires the management of very large datasets
- iii) Coset Enumeration – given a finitely-presented group, and generating words for a subgroup, find the permutation action of the group on the cosets of the subgroup. This is a key technique in almost all computation with finitely-presented groups.
- iv) Simplification of a presentation for a finitely-presented group (Tietze Transformation and/or Knuth-Bendix). This involves subtle string-matching and data structure issues, as well as a very irregular computational structure.

Preliminary work on additional problems may be possible during the course of the project if rapid progress is made on those we have selected. Possible additional problems that we have identified include: subgroup normalizer in permutation groups; basic linear algebra operations over finite fields; and polynomial factorisation.

The milestones for the workpackage are: **M3.1** – completed parallel implementations of each of the chosen algorithms;

and **M3.2** – completed performance measurements for each algorithm. The associated deliverables are: **D3.1** – GAP methods implementing each algorithm in parallel on commodity hardware; and **D3.2** – a report on the suitability of each of the algorithms for parallelisation, including the performance measurements determined by the project. A useful corollary of our focus on key sub-problems is that a number of GAP users will be able to exploit our software directly from within their own applications, without further programming in either GAP or GpH.

6.2 Parallel Application Development Methodology

A primary aim of this project is to develop novel parallel algorithms for computational algebra. This will be greatly assisted by our development platform and methodology. Over a number of years, we have developed a methodology for constructing parallel programs based on systematic refinement of the parallel program [23] in conjunction with high quality performance monitoring tools [15] using high level behavioural abstractions, so-called *evaluation strategies* [36]. By exploiting accurate simulation information geared to a range of abstractions over parallel architectures, programs can be tuned to deliver good performance over a wide variety of parallel architectures [38]. The very high-level nature of our programming environment allows alternative parallel algorithms and behaviours to be tested extremely rapidly, reducing the cost of exploring unprofitable approaches to parallelisation, and avoiding early commitment to a single parallel design. Our system is capable of handling highly irregular computation structures and communication patterns, including arbitrarily nested parallel structures or dynamically phased computations. These features closely match those identified in the computational algebra problems that we propose to study in the project.

6.2.1 The Parallel Compilation System

The sequential GHC compiler supports state-of-the art functional language optimisation for Haskell, delivering performance which may in the best case, when tuned, be within a small constant factor of hand-written C [16], and which is comparable with, or better than, that of the native GAP system. The parallel runtime system adds a small overhead to the one-processor execution time, and can deliver linear/near-linear speedup in the best case. We anticipate that this performance will allow real absolute parallel speedups to be achieved over sequential GAP programs. Performance hotspots can be hand-optimised by rewriting at the algorithm level, by exploiting special language constructs and data structures (for example, strictification and unboxing), or by rewriting critical sections in a lower-level language such as C or assembler. Low-level portability is enhanced by the use of C as a compiler target language, with inter-processor communication built on the widely available PVM or MPI harnesses

6.2.2 Memory Management Issues

Many of the problems we have identified in Section 6.1 rely on complex and irregular data structures that must be communi-

cated between the sequential front-end and parallel back-end. Since both GAP and GpH use similar heap-based automatic memory management mechanisms, with built-in support for “foreign” data structures, memory management and marshalling issues will be correspondingly simplified compared with direct use of e.g. C.

6.2.3 Performance Measurement

An important aspect of the work programme is the systematic development and measurement of the performance of the parallel algorithms implementing each key problem. The performance of each of these will be measured using:

- a good sequential implementation of the algorithm;
- an idealised parallel setting incorporating infinite processors, zero communication and other overhead costs, but realistic instruction cost settings;
- a range of simulations reflecting increasing communication costs, ranging from superscalar on-chip designs to fully distributed systems, using the same communication and overhead costs as for the idealised measurements, and varying the number of processors from 1 to 64;
- realistic simulations for both shared-memory and distributed-memory parallel systems, including realistic communication costs, thread creation and context-switching costs, and instruction costs;
- the actual target architectures.

For consistency and clarity of measurement, and based on expected use patterns, each measured architecture will be homogeneous and assume reliable communications. The initial sequential implementation will provide a strong base-line for subsequent parallel experimentation. The idealised parallel simulations will then serve as the basis for initial tuning of the parallel algorithms, allowing the bounds on potential parallelism to be identified without unnecessary interference from details of the communication architecture etc. Subsequent measurements will clarify the performance on a full range of parallel architectures, and identify the points at which increasing investment in number of processors becomes ineffective in terms of performance gains. The systems will then be tuned for the specific parallel architectures we intend to use, again using highly instrumented simulations, before final performance measurement and tuning on the actual parallel machines. Experience from the measurement process will be used to improve the design of the libraries, algorithms, and perhaps the parallel GpH implementation itself.

7 Relevance to Beneficiaries

The project represents an inter-disciplinary collaboration between pure mathematicians, under the direction of Steve Linton, and computer scientists, under the direction of Kevin Hammond. The work will benefit both bodies: mathematicians through insight into important algorithms and access to new software for running parallel computations; and computer scientists through access to a challenging and novel application base. Modern mathematics, physics and computer

science all present increasing demands for computational support: the response of the numerical computing community is increasingly to move to parallel and distributed computation, as suitable computers, whether in the form of symmetric multi-processors or networks of uniprocessor workstations, become more affordable and easier to use.

This project will enable GAP’s established user base to take advantage of such facilities, through exploiting links to the libraries and algorithms that will be developed by this project, or by exploiting the new parallel programming language constructs in order to produce their own parallel algorithms. The ease of use we expect to achieve is crucial to widespread adoption, since the majority of GAP users have very limited programming expertise.

The computer science community at large will benefit from a clearer understanding of the issues arising in executing non-numerical (symbolic) algorithms in parallel, and in using parallel systems in the context of an extensible system such as GAP, a very different model from conventional batch computing or interactive visualisation. Computer scientists researching parallel systems, and E-science applications will also benefit from the study of an interesting and complex body of real-world problems. The problems involve high-performance irregular, symbolic computations and data: the most difficult class of problem to parallelise, and an area where parallel computing has so far made little impact. Finally, the work will be especially interesting to the functional programming community, forming a new set of problems and applications with close connections to current research issues.

8 Track Record

We are ideally placed to carry out the research for this project, possessing deep and long-standing research expertise in both computational algebra and parallel programming, and having strong research backgrounds in both pure mathematics and computer science. The project brings together primary contributors to two leading research software systems: Steve Linton, representing the GAP computational algebra system; and Kevin Hammond, representing the GpH parallel functional language implementation. In addition, the project has the direct involvement of a world-leading expert in parallel symbolic computation, in the shape of our visiting fellow, Prof. Gene Cooperman (Northeastern University, Boston, USA), plus close ties with the major research institute into symbolic computation at RISC-LINZ, Austria. The two investigators have undertaken significant research in their chosen fields, and also possess considerable expertise in successfully managing research projects, whether funded by the EPSRC or by other research agencies. Both are based within the School of Computer Science at the University of St Andrews, an arrangement which facilitates informal as well as formal collaboration.

8.1 University of St Andrews

The School of Computer Science at the University of St Andrews has a good reputation for its research pioneering into both computational algebra, and programming language design, including functional programming. It received a 4 in the cognate area of Computer Science in the 1996 Research Assessment Exercise. The School is both well housed and equipped, including its own 64-processor parallel machine (funded by an EPSRC JREI grant), which we intend to use in this project.

8.2 Computational Algebra

Dr Steve Linton is a Senior Lecturer in the School of Computer Science and Director of the Centre for Interdisciplinary Research in Computational Algebra (CIRCA) which also involves the School of Mathematics and Statistics. He leads the international consortium that develops GAP, and specifically takes responsibility for the programming language and system interface issues [3] that are directly relevant to this project. Before taking over leadership of the GAP project in 1997, he worked particularly on the kind of large cutting-edge mathematical computations [20, 22, 21] which are likely to be early adopters of the parallel computing possibilities that will be opened up by this grant. At St Andrews, Dr Linton has been Principal Investigator on the following grants:

GR/L21013: Development and Application of GAP, 1996–2000, £183K This grant provided core funding for the GAP project during and following its transfer to the UK, leading to the release of GAP 4. Rated $\alpha/5$ at final review.

The Leverhulme Trust: A Flexible User-interface Architecture for GAP, £72K, 1997–2000 The grants supported experimental work on alternative modes of use for GAP, notable supporting our contribution to the development

of a very novel and successful interactive textbook “Algebra Interactive” [9]

European Commission, OpenMath Project, 1997–2000, £80K at St Andrews, 2.7 MECU overall This grant generally supported the development of the OpenMath standard for representing mathematical information, and a variety of demonstrator applications, including a link to GAP.

He is also a co-investigator on another interdisciplinary project, the recently-announced MathFIT Grant GR/R 29666 which will explore the application of computational group theory to the management of symmetry in constraint solving.

8.3 Parallel Programming

Dr Kevin Hammond is a Lecturer in the School of Computer Science, who has worked extensively in the field of advanced programming language design and implementation, with a focus on parallel functional languages. Before taking up his post at St Andrews he was formerly at Glasgow University, where he undertook research initially as a researcher on an SERC-funded project in association with Profs. Simon Peyton Jones and Phil Wadler, and latterly, independently, as a Royal Society of Edinburgh SOED personal research fellow.

Since receiving his PhD in 1989, he has published widely in the general area of parallel functional programming, producing over 50 books, book chapters, journal papers and other refereed publications focusing on cost issues and performance measurement. He is co-editor (with Greg Michaelson, Heriot-Watt) of the primary reference text in parallel functional programming [14], a 520-page volume published by Springer-Verlag in 1999 and incorporating contributions from over 24 internationally recognised researchers from 7 countries worldwide. The book is designed as a coherent coverage of the subject area, incorporating specially commissioned chapters on general research themes as well as specific current research projects.

On the language design side, Kevin Hammond was a highly active member of the international Haskell design committee, including being joint editor of the Haskell 1.3/1.4 language and library reports [29]. Haskell is the foremost non-strict functional language, it is the subject of 4 text books, and is now in active use at over 200 institutions world-wide, both academic and industrial.

On the implementation side, Kevin Hammond was a major contributor to the development of the highly successful GpH [37] and GHC systems [31], which will be used in this project. His contribution included writing the first ever Haskell compiler, producing the first version of the highly instrumented GranSim parallel simulator for GPH [13], being heavily involved in the design and implementation of the portable parallel GUM implementation [37], and being instrumental in the development of the evaluation strategy approach for abstracting parallel computation [36]. These systems represent the state-of-the-art in functional language implementation, and have been widely used by many researchers world-wide.

In addition to this research experience, which is directly relevant to this project, Kevin Hammond has been involved in the management of a number of research grants either as principal investigator or as co-investigator. He is the sole principal

investigator on EPSRC project GR/L 93379, “Granularity Analysis for parallel functional languages”, was principal investigator on EPSRC grant GR/M 43272, “Distributed Memory Parallel Graph Reduction” (rated as very significant to the field — α 4 — and excellent value for money), and was co-investigator on EPSRC grants GR/M 32351 “Distributed Systems Software” (rated as “tending to outstanding” and excellent value for money) and GR/J 53348, “A Parallel Applicative Database Engine” (given top ratings by EPSRC: highly significant to the field — α 5 — and excellent value for money).

Kevin Hammond has also been highly active in international conference and workshop organisation, serving on a number of programme committees and regularly organising national and international research conventions, as well as being extremely active within the Scottish community. Most recently he has been invited to act as global chair of EuroPar 2002 topic “Parallel Programming, Models, Methods and Programming Languages”, and is programme chair of the 3rd Scottish Functional Programming workshop, held in August 2001.

References

- [1] Amrhein, B., Gloor, O. and W. Kuchlin (1996) *A case study of multithreaded Gröbner basis completion*, in Proc. of ISSAC '96, ACM Press, pp. 95–102.
- [2] Bosma, W., Cannon, J., *Handbook of Magma Functions*, Sydney, 1993.
- [3] T. Breuer and S.A. Linton, The GAP4 Type System: Organizing Algebraic Algorithms. In *ISSAC'98: Proceedings of the 1998 International Symposium on Symbolic and Algebraic Computation*. ACM Press, 1998.
- [4] Bündgen, R., M. Göbel and W. Kuchlin (1994) *Multithreaded AC term re-writing*, in Proc. of PASCO '94, Lecture Notes Series in Computing, World Scientific, vol 5., pp. 84–93.
- [5] Cannon, J., *A general purpose group theory program*, in “Second International Conference on the Theory of Groups, Canberra, 1973” (ed. Newman, M.F.), Lecture Notes in Math. 372, Springer, Berlin, 1974.
- [6] K.C. Chan, A. Díaz, and E. Kaltofen, “A Distributed Approach to Problem Solving in Maple”, *Maple V: Mathematics and its Applications: Proc. 5th Maple Summer Workshop and Symposium*, Boston, 1994, pp. 13–21.
- [7] Char, B. W. *et al*, *Maple V Language Ref. Manual*, Waterloo Maple Publishing, 1991.
- [8] B.W. Char, “A user’s guide to Sugarbush – Parallel Maple through Linda”, *Technical Report MCS-94-01*, Drexel University, Department of Mathematics and Computer Science, 1994.
- [9] A. Cohen, H. Cuypers, and H. Sterk. “Algebra Interactive!”, Springer-Verlag, 1999.
- [10] Comet, S., *On the machine calculation of characters of the symmetric group*, in “Tolfta Skandinaviska Matematikerkongressen” (M. Riesz ed., Lund, 1954. Hakon Ohlssons Boktryckeri, Lund), 18–23.
- [11] Cooperman, G., “Parallel GAP: Mature Interactive Parallel”, in *Groups and computation, III (Columbus, OH, 1999)*, 123–138, de Gruyter, Berlin, 2001;
- [12] T. Gautier and J.L. Roch, “Fast Parallel Algebraic Number Computation”, *Proc. 2nd. Intl. Symp. on Parallel Symbolic Computation (PASCO '97)*, Maui, Hawaii, 1997.
- [13] K. Hammond, H.-W. Loidl and A.S. Partridge, “Improving Granularity for Parallel Functional Programs: a Graphical Winnowing System for Haskell”, *Proc. 1995 Conf. on High Performance Functional Computing (HPFC '95)*, Denver, Co., Apr. 1995, pp. 208–221.
- [14] K. Hammond and G.J. Michaelson (Eds.), *Research Directions in Parallel Functional Programming*, Springer-Verlag, November 1999.
- [15] K. Hammond, D.J. King, H.-W. Loidl, A. Rebon and P.W. Trinder, “The HasPar Suite for Monitoring the Performance of Parallel Haskell Programs”, In Preparation, 2001.
- [16] P. Hartel, M. Feeley, et al. (including K. Hammond), “Pseudoknot: A Floating-Point Intensive Benchmark for Functional Languages”, *Journal of Functional Programming*, **6**(4), July 1996, pp. 621–655.
- [17] Jenks, R.D. and Sutor, R.S., *axiom: The Scientific Computation System*, Springer, 1992.
- [18] Kuchlin, W. (1990) *PARSAC-2: A parallel SAC-2 based on threads* in Proc. AAEECC-8, Lecture Notes in Computer Science, Springer-Verlag, vol. 508, pp. 341–353.
- [19] Laue, R., Neubüser, J., and Shoenwalder, U., *Algorithms for Finite Soluble Groups and the SOGOS system*, in “Computational Group Theory, Durham 1982”, (ed. Atkinson, M.D.), Academic Press, 1984, 105–135.
- [20] S.A. Linton. The art and science of computing in large groups. In *Computational algebra and number theory (Sydney, 1992)*, pages 91–109. Kluwer Acad. Publ., Dordrecht, 1995.
- [21] S.A. Linton, K. Lux, and L.H. Soicher. The primitive distance-transitive representations of the Fischer groups. *Experiment. Math.*, **4**(3):235–253, 1995. ISSN 1058-6458.
- [22] S.A. Linton, R. Parker, P. Walsh, and R. Wilson. Computer construction of the Monster. *J. Group Theory*, **1**(4):307–337, 1998. ISSN 1433-5883.
- [23] H-W. Loidl, P.W.Trinder, C.V. Hall, K. Hammond, S.B. Junaidu, R.G. Morgan and S.L. Peyton Jones, “Large-Scale Parallel Functional Programming”, *Concurrency: Practice & Experience*, December 1999.
- [24] H.-W. Loidl and W. Schreiner, “An interface from Maple to GpH”, in preparation, 2001.
- [25] N. Mannhart and T. Gautier, “Parallelism in Aldor – the Communication Library for Parallel, Distributed Computing”, *Proc. EuroPar'99*, Springer-Verlag LNCS 1685, 1999, pp. 1466–1475.

- [26] Metzner, T., M. Radimersky, A. Sorgatz and S. Wehmeier (1998) *Parallelism in MuPAD 1.4* Technical Report, University of Paderborn Department of Mathematics/Computer Science, available at <http://www.mupad.de/PAPERS/PARALLEL/makrop98.ps.gz>
- [27] Michler, G. O., *High performance computations in group representation theory*, Preprint, Institut für Experimentelle Mathematik, Universität GH Essen, 1998.
- [28] Neubüser, J., Pahlings, H., and Plesken, W., *CAS; design and use of a system for the handling of characters of finite groups*, in “Computational Group Theory, Durham 1982”, (ed. Atkinson, M.D.), Academic Press, 1984, 195–247.
- [29] J.W. Peterson, K. Hammond (eds.), L. Augustsson, B. Boutel, F.W. Burton, J.H. Fasel, A.D. Gordon, R.J.M. Hughes, P. Hudak, T. Johnsson, M.P. Jones, E. Meijer, S.L. Peyton Jones, A. Reid and P.L. Wadler, “Report on the Programming Language Haskell: a Non-Strict Purely Functional Language, Version 1.4”, April 1997.
- [30] *Report on the Non-Strict Functional Language, Haskell* S.L. Peyton Jones (ed.), L. Augustsson, B. Boutel, F.W. Burton, J.H. Fasel, A.D. Gordon, K. Hammond, R.J.M. Hughes, P. Hudak, T. Johnsson, M.P. Jones, E. Meijer, J.C. Peterson, A. Reid, and P.L. Wadler, Yale University, 1999.
- [31] S.L. Peyton Jones, C.V. Hall, K. Hammond, W.D. Par-tain and P.L. Wadler, “The Glasgow Haskell Compiler: a Technical Overview”, *Proc. JFIT '93*, Keele, March 1993.
- [32] D. Petcu, “PVMMaple: A Distributed Approach to Cooperative Work of Maple Processes”, *PVM/MPI*, pp. 216–224, 2000.
- [33] L. Roch and G. Villard. “Parallel Computer Algebra”, *Proc. ISSAC'97*, Preprint IMAG Grenoble France, July 1997.
- [34] The GAP Group, *GAP – Groups, Algorithms, and Programming, Version 4.2*, Aachen, St Andrews, 2000, (<http://www.gap-system.org/gap>).
- [35] Spitznagel, E. L., *Review of Mathematical Software, GAP*, Notices of the AMS, 41 (1994), 780–782.
- [36] P.W. Trinder, K. Hammond, H.-W. Loidl and S.L. Peyton Jones, “Algorithm + Strategy = Parallelism”, *Journal of Functional Programming*, **8**(1), Jan. 1998, pp. 23–60.
- [37] P.W. Trinder, K. Hammond, J.S. Mattson Jr., A.S. Partridge and S.L. Peyton Jones, “GUM: a Portable Parallel Implementation of Haskell”, *Proc. 1996 ACM. Conf. on Programming Language Design and Implementation (PLDI '96)*, Philadelphia, May 1996, pp. 79–90.
- [38] P.W. Trinder, H.-W. Loidl, E. Barry Jr., K. Hammond, U. Klusik, S.L. Peyton Jones, and A.J. Rebón Portillo, “The Multi-Architecture Performance of the Parallel Functional Language GPH”, *Proc. Euro-Par 2000*, Munich, Germany, Springer-Verlag LNCS, August 2000.
- [39] S. Wolfram, *The Mathematica book*, Fourth edition, Wolfram Media, Inc., Champaign, IL, 1999