SymGrid: a Framework for Symbolic Computation on the Grid

Kevin Hammond[1], Abdallah Al Zain[2], Gene Cooperman[3], Dana Petcu[4], and Phil Trinder[2]

[1] School of Computer Science, University of St Andrews, St Andrews, UK.
email: kh@cs.st-and.ac.uk

[2] Dept. of Mathematics and Comp. Sci., Heriot-Watt University, Edinburgh, UK.
email: {ceeatia,trinder}@macs.hw.ac.uk

[3] College of Computer Science, Northeastern University, Boston, USA.
email: gene@ccs.neu.edu

[4] Institute e-Austria, Timişoara, Romania. **email:** petcu@info.uvt.ro

Abstract. This paper introduces the design of **SymGrid**, a new Grid framework that will, for the first time, allow multiple invocations of symbolic computing applications to interact via the Grid. **SymGrid** is designed to support the specific needs of symbolic computation, including computational steering (greater interactivity), complex data structures, and domain-specific computational patterns (for irregular parallelism). A key issue is heterogeneity: **SymGrid** is designed to orchestrate components from different symbolic systems into a single coherent (possibly parallel) Grid application, building on the **OpenMath** standard for data exchange between mathematically-oriented applications. The work is being developed as part of a major EU infrastructure project.

1 Introduction

Symbolic Computation is often distinguished from Numerical Analysis by the observation that symbolic computation deals with exact computations, while numerical analysis deals with approximate quantities, including issues of floating point error. Examples of problems that motivated the original development of symbolic computation include symbolic differentiation, indefinite integration, polynomial factorization, simplification of algebraic expressions, and power series expansions. Because of the focus on exactness, symbolic computation has followed a very different evolutionary path from that taken by numerical analysis. In contrast to notations for numerical computations, which have emphasised floating point arithmetic, monolithic arrays, and programmer-controlled memory allocation, symbolic computing has emphasized functional notations, greater interactivity, very high level programming abstractions, complex data structures, automatic memory management etc. With this different direction, it is not surprising that symbolic computation has different requirements from the Grid than

traditional numerical computations. For example, the emphasis on interactivity for symbolic computation leads to a greater stress on *steering* of computations, and the emphasis on functional programming has led to the identification of sophisticated *higher-order* computational patterns. Similar patterns have recently found commercial use in, for example, Google's **MapReduce** system [22, 23].

In order to support the specific needs of symbolic computation on the Grid, as part of the EU Framework VI SCIEnce project (Symbolic Computation Infrastructure in Europe, RII3-CT-2005-026133), we have designed a new Grid framework, **SymGrid**, which builds on and extends standard Globus middleware capabilities, providing support for Grid Services and for orchestration of symbolic components into high-performance Grid-enabled applications forming a computational Grid. The project will initially integrate four major oomputational algebra systems into **SymGrid**: Maple [14], GAP [25], Kant [21] and MuPad [28]. In this way, heterogenous symbolic components can be composed into a single large-scale (possibly parallel) application. Our work builds on earlier, long-standing work on adaptive parallel systems [7], parallel symbolic computations [17, 18] and distributed/Grid-enabled symbolic computations [30].

1.1 Novelty

The key novelties of the **SymGrid** approach are:

1. we integrate several represenatative symbolic computation systems (Maple [14], GAP [25], Kant [21] and MuPad [28]) into a single, generic and non-exclusive middleware framework;
2. we provide a sophisticated interactive *computational steering* interface integrating seamlessly into the interactive front-ends provided by each of our target symbolic computation systems, and providing simple, transparent and high-level access to Grid services;
3. by defining common data and task interfaces for all systems, we allow complex computations to be constructed by orchestrating heterogeneous distributed components into a single symbolic application;
4. by exploiting well-established adaptive middleware that we have developed to manage complex irregular parallel computations on clusters and shared-memory parallel machines, and that has recently been ported to the Grid, we allow a number of advanced autonomic features that are important to symbolic computations including automatic control of task granularity, dynamic task creation, implicit asynchronous communication, automatic sharing-preserving data-marshalling and unmarshalling, ultra-lightweight work stealing and task migration, virtual shared memory, and distributed garbage collection;
5. we identify new domain-specific patterns of symbolic computation that may be exploited to yield platform-specific implementations for a wide range of classical symbolic computations, and which may be combined dynamically to yield complex and irregular parallel computation structures; and

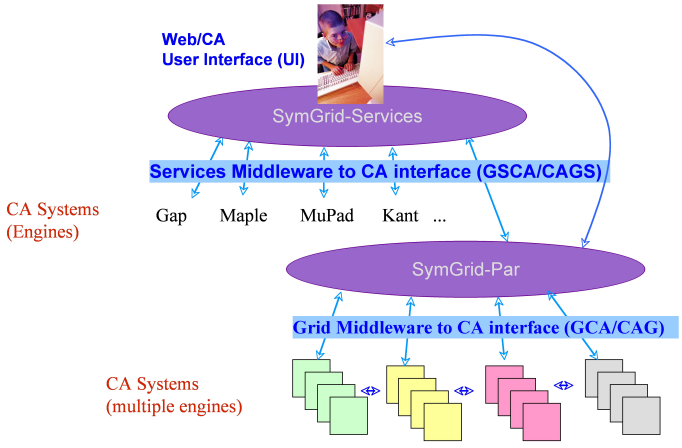

Fig. 1. The SymGrid Design

6. we target a new user community which may have massive computational demands yet where exposure to parallelism/Grids is not common; this may serve as a template for extending Grid technologies to similar user bases.

This paper introduces the design of the **SymGrid** framework (Section 2), describes middleware components that support both Grid services (Section 2.1) and the composition of heterogeneous symbolic components into large-scale Grid-enabled applications (Section 2.2), identifies new and important computational patterns that are specific to symbolic computation (Section 3), and places our work in the context of previous work on parallel and Grid-enabled symbolic computations (Section 4).

2 The SymGrid Design

SymGrid comprises two main middleware components (Figure 1): **SymGrid-Services** provides a generic interface to Grid services, which may be engines of computational algebra packages; **SymGrid-Par** links multiple instances of these engines into a coherent parallel application that may be distributed across a geographically-distributed computational Grid. **SymGrid-Par** may itself be registered as a Grid service. The engines are linked by well-defined interfaces that use the standard **OpenMath** protocol for describing mathematical objects [5], developed as part of EU Framework V project MONET (IST-2001-34145). We have constructed prototype implementations of both components and are in the process of robustifying these for general use.

2.1 SymGrid-Services

Modern Grid and Web technologies allow simple access to remote applications and other services. Services are exposed through standard ports, which provide mechanisms to allow the discovery of new services and to support interaction with those services. **SymGrid-Services** provides a set of WSRF-compliant interfaces from symbolic computations to both Grid and Web services, and allows straightforward encapsulation of symbolic computations as Grid service components, including automatic client generation. Mathematical objects are communicated between users and services using **OpenMath** protocols.

While Grid services have different goals from pure Web services (sharing computing power and resources such as disk storage databases and software applications, compared with simply sharing information), a Grid service is essentially a Web service with a few new additions: stateful services, service instantiation, named service instances, two-level naming scheme, a base set of service capabilities, and lifetime management. These new capabilities improve user interaction with remote symbolic computing services: prior computations can be stored in the service state, using instantiation, personalized services can be created, the naming schemes allow services to be easily modified, standard service data elements allow standard search facilities to be implemented, and, due to the transient character of the services, resource management can easily be performed . All of these features are supported through **SymGrid-Services**.

2.2 SymGrid-Par

SymGrid-Par orchestrates symbolic components into a (possibly parallel) Grid-enabled application. Each component executes within an instance of a Grid-enabled engine, which can be geographically distributed to form a wide-area computational Grid, built as a loosely-coupled collection of Grid-enabled clusters. Components communicate using the same **OpenMath** data-exchange protocol that is also used by **SymGrid-Services**. In this way, a high degree of integration is achieved between services and high-performance computations.

SymGrid-Par is built around **GRID-GUM** [7], a system designed to support parallel computations on the Grid, and which has been adapted to interface with symbolic engines. **GRID-GUM** builds on the basic MPICH-G2 transport protocols and task management capabilities to provide a range of very high-level facilities aimed at supporting large-scale, complex parallel systems. It includes support for ultra-lightweight thread creation, distributed virtual shared-memory management, multi-level scheduling support, automatic thread placement, automatic datatype-specific marshalling/unmarshalling, implicit communication, load-based thread throttling, and thread migration. It thus provides a flexible, adaptive, *autonomic* environment for managing parallelism/distribution at various degrees of granularity.

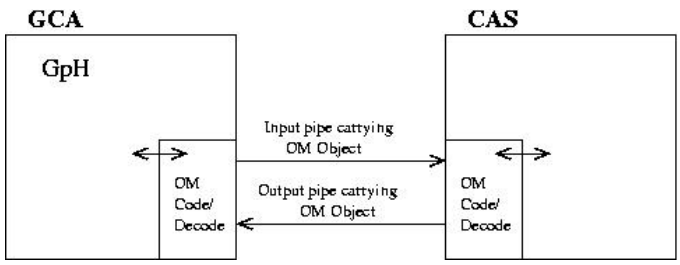

Fig. 2. GCA Design

SymGrid-Par comprises two interfaces: CAG links the computational algebra systems (CASs) of interest to **GRID-GUM**; and GCA (Figure 2) conversely links **GRID-GUM** to these CASs. The CAG interface is used by the CAS to interact with **GRID-GUM**. **GRID-GUM** then uses the GCA interface to invoke remote CAS functions and communicate with the CASs etc. In this way, we achieve a clear separation of concerns: **GRID-GUM** deals with issues of thread creation/coordination and orchestrates the CAS engines to work on the application as a whole; while the CAS engine deals solely with execution of individual algebraic computations. Together, the systems provide a powerful, but easy-to-use, framework for executing heterogeneous symbolic computations on a computational Grid. We exploit this framework to provide support for commonly found patterns of computation that may be used directly from within symbolic programs.

3 Patterns of Symbolic Computation

We have identified a number of common patterns used in symbolic computation applications. Each of these patterns is potentially amenable to specific kinds of parallel execution. **SymGrid** will support these patterns as dynamic *algorithmic skeletons* [16], which may be called directly from within the computational steering interface. In general (and unlike most previous skeletons approaches), these patterns may be nested or composed dynamically as required to form the Grid computation. They may also mix computations taken from different computational algebra systems (in which case the computations are steered to an appropriate engine using **SymGrid-Par**).

3.1 Standard Patterns

The standard patterns that we have identified are listed below. These patterns are a subset of those identified by Gorlatch and Cole as appropriate algorithmic

skeletons for the Grid [8, 15], and are also similar to those previously identified for the **ParGap** parallel implementation of GAP [19].

```
parMap::       (a->b) ->                    [a] ->   [b]
parZipWith::   (a->b->c) -> [a] ->          [b] ->   [c]
parReduce::    (a->b->b) ->  b ->           [a] ->    b
parMapReduce:: (a->b->b) -> (c->[(d,a)]) -> c  ->   [(d,b)]
```

Here each argument to the pattern is separated by an arrow (`->`), and may operate over lists of values (`[..]`), or pairs of values (`(..,..)`). All of the patterns are *polymorphic*: `a`, `b` etc. stand for (possibly different) concrete types. The first argument in each case is a function of either one or two arguments that is to be applied in parallel. For example, `parMap` applies its function argument to each element of its second argument (a list) in parallel, and `parReduce` will reduce its third argument (a list) by applying the function between pairs of elements, ending with the value supplied as its second argument. The parallel semantics and implementations of these standard patterns are well established [19, 8, 15] and we will therefore not describe these in detail here.

3.2 Domain-Specific Patterns for Symbolic Computation

Parallel symbolic computation often requires irregular parallelism, which in turn leads to non-traditional patterns of parallel behaviour, Based on our experience with parallel symbolic computations, we can identify a number of new *domain-specific* patterns that may arise in a variety of applications. These include:

1. *orbit calculation:* given an open queue of new states, generate neighbors and place them in the open queue if they are not already present (related to breadth-first search).
2. *duplicate elimination:* given two lists, merge them while eliminating duplicates. Typical implementations include: (i) sort and merge; (ii) hash one list and check if the other list contains a duplicate
3. *completion algorithm:* given a set of objects, new objects can be *generated* from any pair of objects. Each new object is *reduced* against existing objects according to some rules, and if an object is thereby reduced to the trivial object, it is discarded. The newly reduced object is then used to reduce other existing objects.
4. *chain reduction:* given a chain of objects, (for example, the rows of a matrix during a Gaussian elimination), any later object in the chain must be reduced against each of the earlier objects in the chain. The algorithm terminates when no further reductions are possible.
5. *partition backtracking:* given a set of objects such as permutations acting on a set of points the algorithm searches for some *basis objects*. Any known basis objects allow additional pruning during the algorithm. As an optimization to backtracking search, the set of points may be partitioned into a disjoint

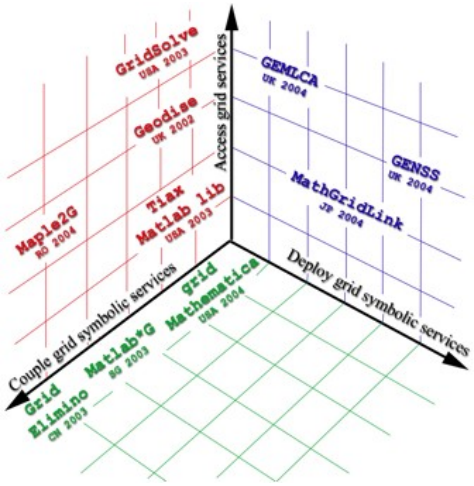

Fig. 3. Existing Grid-enabled Symbolic Computing Systems

union of two subsets, and the algorithm then finds all basis objects that respect this partition. The latter objects allow the remaining search to be more aggressively pruned.

Each of these patterns gives rise to highly-parallel computations with a high degree of inter-task interactions, and complex patterns of parallelism that may involve a mixture of task- and data-parallel computations.

4 Related Work

4.1 Parallel Symbolic Computation

Work on parallel symbolic computation dates back to at least the early 1990s. Roch and Villard [31] provide a good general survey as of 1997. Significant research has been undertaken for specific parallel computational algebra algorithms, notably term re-writing and Gröbner basis completion (e.g. [9, 11]). A number of one-off parallel programs have also been developed for specific algebraic computations, mainly in representation theory [27]. While several symbolic computation systems include some form of operator to introduce parallelism (e.g. parallel GCL, which supports Maxima [17], parallel Maple [10], or parallel GAP [18]), there does not exist a large corpus of production parallel algorithms. This is at least partly due to the complexities involved in programming such algorithms using explicit parallelism, and the lack of generalised support for

communication, distribution etc. By abstracting over such issues, and, especially, through the identification of the domain-specific patterns noted above, we anticipate that **SymGrid** will considerably simplify the construction of such computations.

4.2 Symbolic Computation for Computational Grids

While, as we have outlined above, parallel symbolic systems exist that are suitable for either shared-memory or distributed memory parallel systems (e.g. [13, 12, 19, 26, 4]), there is relatively little work on Grid-based symbolic computation. Figure 3 shows the relationship between the main existing systems including our own **Maple2G** system [30]. While the vision of Grid computing is that of simple and low-cost access to computing resources without artificial barriers of physical location or ownership, none of these Grid-enabled systems conforms entirely to the basic requirements of this vision. **SymGrid** is therefore highly novel in aiming both to allow the construction of heterogeneous symbolic computations and in allowing seamless access to the Grid from within symbolic systems. In order to achieve such a vision, we must be able to:

1. *deploy* symbolic Grid services;
2. allow *access* to available Grid services from the symbolic computation; and,
3. *combine* different Grid symbolic services into a coherent whole.

This is the challenge that we have attempted to meet in the design of the **SymGrid** system.

4.3 Symbolic Computations as Grid Services

Even less work work has so far been carried out to interface CASs to the Grid. A number of projects have considered the provision of CASs as Grid services, e.g. GENSS [2], GEMLCA [24], Netsolve/GridSolve [6], Geodise [3], MathGridLink [32], Maple2G [30], GridMathematica [1]. More details are given in [29]. Using the above mentioned software prototypes or tools, Grid services can be called from within CASs. Still there has been very little work on adapting CASs so that they can cooperate as part of a general Grid resource. None of these systems is, however, capable of linking heterogeneous CASs as in **SymGrid**.

5 Conclusions and Further Work

We have introduced **SymGrid**, a new middleware framework supporting generic symbolic computations on computational/collaborative Grids. Prototype implementations of the two main **SymGrid** middleware components, **SymGrid-Services** and **SymGrid-Par**, have been based on earlier successful but system-specific work. We have successfully tested these on a number of computations

running on the Grid, including a parallel orbit calculation, which has delivered promising speedup results on a test Grid linking the UK, Germany and Romania. We are in the process of enhancing these implementations to provide improved robustness, full heterogeneity and good support for the patterns of computation that we have identified above. In collaboration with our research partners, we are also identifying good and realistic symbolic computation exemplars, including ones with novel heterogeneous aspects.

While we have focused in this paper on the four computational algebra systems that form the immediate target of the SCIEnce project (Maple, Gap, Kant and MuPad), the majority of our work is highly generic, and we intend in due course to explore the inclusion of other symbolic systems. In the longer term, we also intend to work on providing highly autonomic scheduling and work management resource managers that will take account of information about future execution behaviours of the parallel computation. Unlike many traditional (often numeric) Grid computations, which have a regular (and therefore relatively easily predicted behaviour), the dynamic and highly irregular nature of the workload means that we must explore new approaches based, for example, on statistical prediction or static analysis. We will also explore a new lightweight mechanism for supporting automatic task recovery to improve fault tolerance for long running computations [20], which has not yet been applied in a Grid setting. This will provide us with increased robustness in (potentially) unreliable settings, such as widely-distributed computational Grids built on standard network communication protocols.

References

1. GridMathematica2, `http://www.wolfram.com/products/gridmathematica/`.
2. GENSS, `http://genss.cs.bath.ac.uk/index.htm`, 2007.
3. Geodise, `http://www.geodise.org/`, 2007.
4. The GpH-Maple Interface, `http://www.risc.uni-linz.ac.at/software/ghc-maple/`, 2007.
5. The OpenMath Format,`http://www.openmath.org/`, 2007.
6. S. Agrawal, J. Dongarra, K. Seymour, and S. Vadhiyar. NetSolve: past, present, and future; a look at a Grid enabled server. In *Making the Global Infrastructure a Reality*, pages 613–622. Wiley, 2003.
7. A. Al Zain, P. Trinder, H.-W. Loidl, and G. Michaelson. Managing Heterogeneity in a Grid Parallel Haskell. *J. Scalable Comp.: Practice and Experience*, 6(4), 2006.
8. M. Alt, J. Dünnweber, J. Müller, and S. Gorlatch. HOCs: Higher-Order Components for grids. In T. Getov and T. Kielmann, editors, *Components Models and Systems for Grid Applications*, CoreGRID, pages 157–166, 2004.
9. B. Amrheim, O. Gloor, and W. Küchlin. A case study of multithreaded Gröbner basis completion. In *In Proc. of ISSAC'96*, pages 95–102. ACM Press, 1996.
10. L. Bernardin. Maple on a massively parallel, distributed memory machine. In *Proc. PASCO '97*, pages 217–222. ACM Press, 1997.
11. R. Bündgen, M. Göbel, and W. Küchlin. Multi-threaded AC term re-writing. In *Proc. PASCO'94*, volume 5, pages 84–93. World Scientific, 1994.

12. K.C. Chan, A. Díaz, and E. Kaltofen. A Distributed Approach to Problem Solving in Maple. In *Proc. 5th Maple Summer Workshop and Symp.*, pages 13–21, 1994.
13. B.W. Char. A user's guide to Sugarbush — Parallel Maple through Linda. Technical report, Drexel University, Dept. of Mathematics and Comp. Sci., 1994.
14. B. W. Char et al. *Maple V Language Reference Manual*. Maple Publishing, Waterloo Canada, 1991.
15. M.I. Cole. *Algorithmic Skeletons: Structured Management of Parallel Computation*. The MIT Press, Cambridge, MA, 1989.
16. M.I. Cole. Algorithmic Skeletons. In K. Hammond and G. Michaelson, editors, *Research Directions in Parallel Functional Programming*, chapter 13, pages 289–304. Springer-Verlag, 1999.
17. G. Cooperman. STAR/MPI: Binding a parallel library to interactive symbolic algebra systems. In *Proc. Intl. Symp. on Symbolic and Algebraic Computation (ISSAC '95)*, volume 249 of *Lecture Notes in Control and Information Sciences*, pages 126–132. ACM Press, 1995.
18. G. Cooperman. GAP/MPI: Facilitating parallelism. In *Proc. DIMACS Workshop on Groups and Computation II*, volume 28 of *DIMACS Series in Discrete Maths. and Theoretical Comp. Sci.*, pages 69–84. AMS, 1997.
19. G. Cooperman. Parallel GAP: Mature interactive parallel. *Groups and computation, III (Columbus, OH,1999)*, 2001. de Gruyter, Berlin.
20. G. Cooperman, J. Ansel, and X. Ma. Transparent adaptive library-based checkpointing for master-worker style parallelism. In *Proc. IEEE Intl. Symp. on Cluster Computing and the Grid (CCGrid06)*, pages 283–291, 2006.
21. M. Daberkow, C. Fieker, J. Klüners, M. Pohst, K. Roegner, M. Schörnig, and K. Wildanger. Kant v4. *J. Symb. Comput.*, 24(3/4):267–283, 1997.
22. J. Dean. Experiences with MapReduce, an abstraction for large-scale computation. In *Proc. PACT 2006 – Intl. Conf. on Parallel Architectures and Compilation Techniques*, page 1, 2006.
23. J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proc. OSDI '04 – Sixth Symp. on Operating System Design and Implementation*, pages 137–150, 2004.
24. T. Delaitre, A. Goyeneche, P. Kacsuk, T. Kiss, G.Z. Terstyanszky, and S.C. Winter. GEMLCA: Grid Execution Management for Legacy Code Architecture Design. In *Proc. 30th EUROMICRO Conference*, pages 305–315, 2004.
25. The GAP Group. Gap – groups, algorithms, and programming, version 4.2, 2000. St Andrews, http://www.gap-system.org/gap.
26. W. Küchlin. PARSAC-2: A parallel SAC-2 based on threads. In *Proc. AAECC-8: Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, Tokyo, LNCS 508*, pages 341–353. Springer-Verlag, 1990.
27. G. O. Michler. *High performance computations in group representation theory*. Preprint, Institut fur Experimentelle Mathematik, Univerisitat GH Essen,, 1998.
28. K. Morisse and A. Kemper. The Computer Algebra System MuPAD. *Euromath Bulletin*, 1(2):95–102, 1994.
29. D. Petcu, D. ¸Tepeneu, M. Paprzycki, and Ida T. In *Engineering the Grid: status and perspective*, pages 91–107. American Scientific Publishers, 2006.
30. D. Petcu, M. Paprycki, and D. Dubu. Design and Implementation of a Grid Extension of Maple. *Scientific Programming*, 13(2):137–149, 2005.
31. L. Roch and G. Villard. Parallel computer algebra. In *ISSAC'97*. Preprint IMAG Grenoble France, 1997.
32. D. Tepeneu and T. Ida. MathGridLink – Connecting Mathematica to the Grid. In *Proc. IMS '04, Banff, Alberta*, 2004.