SymGrid-Par: Designing a Framework for Executing Computational Algebra Systems on Computational Grids

Abdallah Al Zain[1], Kevin Hammond[2], Phil Trinder[1], Steve Linton[2], Hans-Wolfgang Loidl[3], and Marco Costanti[2]

[1] Dept. of Mathematics and Comp. Sci., Heriot-Watt University, Edinburgh, UK
{ceeatia,trinder}@macs.hw.ac.uk
[2] School of Computer Science, University of St Andrews, St Andrews, UK
{kh,sal,costanti}@cs.st-and.ac.uk
[3] Ludwig-Maximilians Universität, München, Germany
hwloidl@informatik.uni-muenchen.de

Abstract. SymGrid-Par is a new framework for executing large computer algebra problems on computational Grids. We present the design of **SymGrid-Par**, which supports multiple computer algebra packages, and hence provides the novel possibility of composing a system using components from different packages. Orchestration of the components on the Grid is provided by a Grid-enabled parallel Haskell (GpH). We present a prototype implementation of a core component of **SymGrid-Par**, together with promising measurements of two programs on a modest Grid to demonstrate the feasibility of our approach.

1 Introduction

This paper considers the design of high-performance parallel computational algebra systems targeting computational Grids, undertaken as part of the European Union Framework VI grant RII3-CT-2005-026133 (SCIEnce). Parallelising computational algebra problems is challenging since they frequently possess highly-irregular data- and computational-structures. We describe early stages of work on the **SymGrid** system that will ultimately Grid-enable a range of important computational algebra systems, including at least Maple [12], MuPad [21], Kant [16] and GAP [18]. The **SymGrid** system comprises two distinct parts: **SymGrid-Services** allows symbolic computations to access, and to be offered as, Grid services; conversely, **SymGrid-Par** enables the parallel execution of large symbolic computations on computational Grids. This paper focuses on **SymGrid-Par**. While there are some parallel symbolic systems that are suitable for either shared-memory or distributed memory parallel systems (e.g. [13,11,15,19,5]), work on Grid-based symbolic systems is still nascent, and our work is therefore highly novel, notably in aiming to allow the construction

Y. Shi et al. (Eds.): ICCS 2007, Part II, LNCS 4488, pp. 617–624, 2007.

of heterogeneous computations, combining components from different computational algebra systems. In this paper, we introduce the design of **SymGrid-Par** (Section 2); outline a prototype implementation (Section 3) and present some preliminary results to demonstrate the realisability of our approach (Section 4). In particular, we demonstrate the integration of GRID-GUM with one important computational algebra system, the GAP system for programming with groups and permutations, and show that we can exploit parallelism within a single Grid-enabled cluster. This represents the first step towards a general heterogeneous framework for symbolic computing on the Grid that will eventually allow the orchestration of complete symbolic applications from mixed components written using different computational algebra systems, running on a variety of computer architectures in a geographically dispersed setting, and accessing distributed data and other resources.

2 The SymGrid-Par Middleware Design

Computational algebra has played an important role in notable mathematical developments, for example in the classification of Finite Simple Groups. It is essential in several areas of mathematics which apply to computer science, such as formal languages, coding theory, or cryptography. Applications are typically characterised by complex and expensive computations that would benefit from parallel computation, but which may exhibit a high degree of irregularity in terms of both data- and computational structures. In order to provide proper support for high-performance symbolic computing applications, we therefore use a multi-level approach where parallelism may be exploited within a local cluster (or, indeed, within a single multiprocessor/multicore system), and where individual clusters may then be marshalled into a larger heterogeneous system. In order to allow adequate flexibility, we also provide support for dynamic task allocation, rebalancing and migration. Although we will not discuss it in this paper, our design also allows us to exploit the availability of specific Grid resources, which may not be distributed across the entire computational Grid.

The **SymGrid-Par** middleware is built on the GRID-GUM [8] Grid-enabled implementation of Glasgow Parallel Haskell (GPH) [25], a well-established semi-implicitly parallel extension to the standard Glasgow Haskell Compiler. GPH provides various high-level parallelism services including support for ultra-light-weight threads, virtual shared-memory management, 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 environment for managing parallelism at various degrees of granularity, and has therefore been ported to a variety of shared-memory and distributed-memory systems. GRID-GUM replaces the MPI-based low-level communication library in GPH with MPICH-G2, which integrates with standard Globus Toolkit middleware.

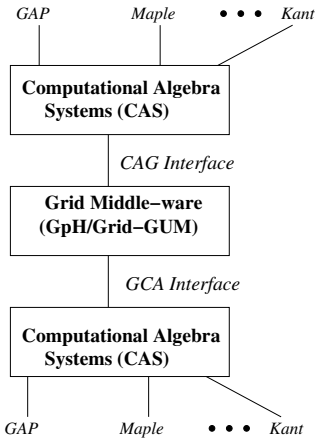

Fig. 1. SymGrid-Par Design Overview

```
<OMOBJ> <OMA>
<OMS name=''FuncName'' cd=''SomeCD'' />
<OMV name=''arg 1'' />
  •
  •
<OMV name=''argn N'' />
</OMA>
</OMOBJ>
```

Fig. 2. GCA OpenMath Service Request

Our overall design is shown in Figure 1. **SymGrid-Par** comprises two interfaces: CAG links computational algebra systems (CASs) to GpH; and GCA conversely links GpH to these CASs. The purpose of the CAG/GCA interfaces is to enable CASs to execute on computational Grids, e.g. on a loosely-coupled collection of Grid-enabled clusters. This is achieved by calling from the CAS to the Grid-enabled GpH middleware using CAG. This, in turn, calls CAS functions on remote processing elements using GCA.

2.1 GCA Design

The purpose of the GCA interface is to allow CAS functions to be invoked from GpH. In this way, GpH deals with issues of process creation/coordination, and the CAS deals with the algebraic computations. GpH and the CAS run as separate operating system processes, communicating through shared pipes. Figure 3 shows the design of the GCA interface. Unidirectional pipes connect each GRID-GUM process to the companion CAS process, as shown in Figure 4. Objects that are communicated between the two systems are encoded using the standard OpenMath [6] format for exchanging mathematical data (see Figure 2).

2.2 CAG Design

The CAG interface will support low-effort Grid programming by providing *algorithmic skeletons* [14] that have been tailored to the needs of computational Grids. Figure 5 shows the standard GAP functions that we believe can form the basis for an initial set of skeletons (ParGAP [15] has also identified a similar set of parallel operations). Here each argument to the pattern is separated by an arrow (`->`), and may operate over lists of values (`[..]`), or pairs of values

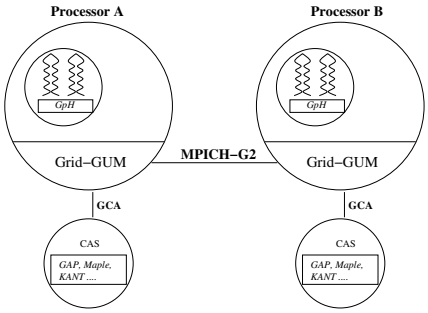

Fig. 3. GCA Interface

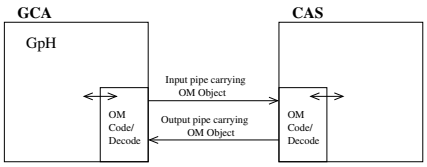

Fig. 4. GCA Design

(`..,..`)). 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.

3 The GCA Prototype

The GCA prototype (Figure 6) interfaces GpH with GAP, connecting to a small interpreter that allows the invocation of arbitrary GAP functions, marshalling/unmarshalling data as required. The interface consists of both C and Haskell fragments. The C component is mainly used to invoke operating system services that are needed to initiate the GAP process, to establish the pipes, and to send and receive commands/results from GAP process. It also provides support for static memory that can be used to maintain state between calls.

The functions *GAPEval* and *GAPEvalN* allow GpH programs to invoke GAP functions by simply giving the function name and a list of its parameters as

```
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)]
```

Fig. 5. CAG Functions

```
GAPEval :: String –> [GAPObject] –> GAPObject

GAPEvalN :: String –> [GAPObject] –> [GAPObject]

String2GAPExpr :: String –> GAPObject

GAPExpr2String :: GAPObject –> String
```

Fig. 6. GCA Prototype Functions

```
smallGroupSearch :: Int -> Int -> Int ->
        ((Int,Int) -> (Int,Int,Bool)) -> [(Int,Int)]
smallGroupSearch lo hi chunkSize pred =
        concat (map (ifmatch pred) [lo..hi] `using`
              parListChunk chunkSize)

ifmatch :: ((Int,Int) -> (Int,Int,Bool)) -> Int -> [(Int,Int)]
ifmatch predSmallGroup n = [ (i,n) | (i,n,b) <-
         ((map predSmallGroup [(i,n) | i <-
           [1..nrSmallGroups n]]) `using`
            parListBigChunk 10000), b]

predSmallGroup :: (Int,Int) -> (Int,Int,Bool)
predSmallGroup (i,n) =
        (i,n,(gapObject2String (gapEval "IntAvgOrder"
         [int2GAPObject n, int2GAPObject i]))== "true")

nrSmallGroups :: Int -> Int
nrSmallGroups n = gapObject2Int (gapEval
                        "NrSmallGroups" [int2GAPObject n])
```

Fig. 7. **smallGroup** Kernel

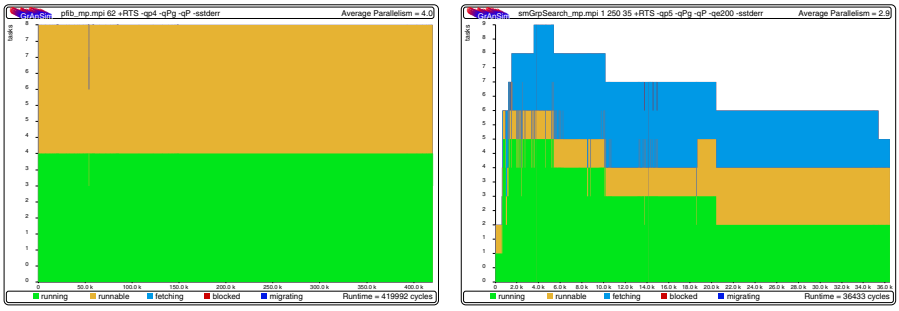

Fig. 8. lift: Activity Profile for (**parFib** 62) on 4 PEs; **right:** Activity Profile for **smallGroup** (1-250) on 5 PEs

*GAPObject*s. The *GAPEvalN* function is used to invoke GAP functions returning more than one object. Finally, *String2GAPExpr* and *GAPExpr2String* convert GAP objects to/from our internal Haskell format.

4 Preliminary GCA Prototype Performance Results

We have measured the performance of our GCA Prototype using two simple programs: **parFib**, a parallel benchmark that is capable of introducing large quantities of very fine-grained parallelism; and **smallGroup**, a group-algebra program that exhibits highly-irregular data-parallelism. In both cases GpH uses GCA to invoke the GAP engine to perform actual computations, dealing only with the decomposition of the problem, marshalling etc. The kernel of the **smallGroup** program is shown in Figure 7. Invocations of *GAPEval* can be clearly seen in

predSmallGroup and *nrSmallGroups*, and the associated marshalling of arguments and results is also clearly visible. Our experiments were performed on a Beowulf-style cluster of workstations at Heriot-Watt University, where each processing element (PE) is a 3GHz Celeron processor with 500kB cache, running Linux Fedora Core-2. All PEs are connected through a 100Mb/s fast Ethernet switch with a latency of 142μs, as measured under PVM 3.4.2. In the following results, each runtime is shown as the median of three execution times.

For (**parFib** 62), the GCA prototype implementation delivers good parallel performance on four processors, requiring 539s in the parallel case, as opposed to a sequential time of 2,559s (a superlinear speedup of 4.75, with average parallelism of 4.0). Figure 3 (above) shows the corresponding GpH activity profile, where time on the x-axis is plotted against the number of threads in various states, e.g. running or blocked, on the y-axis. Note that the activity profiles record only GpH computations, and do not expose activity in the underlying GAP process, which may be lower or higher. The corresponding results for **smallGroup** show that on five PEs, the computation of groups in the interval between 1 and 250 is completed in 37 seconds, compared with a sequential time of 144s, that is a speedup of 3.9 at an average parallelism of 2.9. Figure 8 shows the corresponding activity profile for the GpH component. In order to estimate overheads due to parallelisation, we have also measured sequential times for this problem using the standard GAP sytem. In this case, the problem was executed in 87s, that is the cost of marshalling/unmarshalling data and context switching between processes amounts to 66%. While this is a relatively high figure, we anticipate that we should be able to reduce this cost by careful tuning of the **GRID-GUM** runtime system. While these represent very early results for **SymGrid-Par**, and we therefore now intend to explore performance both on larger numbers of processors and on multiple clusters, and for larger-scale applications, it is clear that *real benefit* can be obtained for computational algebra problems on a clustered system without major rewriting of the computational algebra system.

5 Related Work

Significant research has been undertaken for specific parallel computational algebra algorithms, notably term re-writing and Gröbner basis completion (e.g. [9, 10]). A number of one-off parallel programs have also been developed for specific algebraic computations, mainly in representation theory [2]. There is, however, little if any support for parallelism in the most widely-used CASs such as Maple, Axiom or GAP. As research systems, Maple/Linda-Sugarbush [13] supports sparse modular gcd and parallel bignum systems, with Maple/DSC [11] supporting sparse linear algebra, and ParGAP [15] supporting very coarse-grained computation between multiple GAP processes. There have also been a few attempts to link parallel functional programming languages with computer algebra systems, for example, the GHC-Maple interface [5]; or the Eden-Maple system [20]. None of these systems is in widespread use at present, however, and none supports the broad range of computational algebra applications we intend to target.

Roch and Villard [23] provide a good general survey of work in the field as of 1997. Even less work has so far been carried out to interface CASs to the Grid. While a number of projects have considered the provision of CASs as Grid services, often extending existing web services frameworks, e.g. GENSS [3], GEMLCA [17], Netsolve/GridSolve [7], Geodise [4], MathGridLink [24] or Maple2G [22], and systems such as GridMathematica [1] allow Grid services to be called from within CASs, there has been very little work on adapting CASs so that they can cooperate as part of a general Grid resource. Key work is in the Maple2G system that is now being developed as part of our SCIEnce project. None of these systems is, however, capable of linking heterogeneous CASs as in **SymGrid**.

6 Conclusions

This paper has presented the design of **SymGrid-Par**, a framework for symbolic computing on heterogeneous computational Grids that uniquely enables the construction of complex systems by composing components taken from different symbolic computing systems. The core of **SymGrid-Par** is a pair of standard interfaces (CAG and GCA) that interface between the symbolic computing systems and the GpH middleware. We have discussed a prototype GCA implementation and reported promising performance measurements for two simple GAP/GpH programs. In ongoing work funded by the EU FP VI SCIence project, we now intend to implement the more sophisticated CAG and GCA interfaces, initially for GAP, and then for Kant, Maple and MuPad. The implementations will be validated on larger symbolic computations and robustified. We also plan to demonstrate the inter-operation of multiple symbolic computing systems. In the longer term we will investigate issues associated with scheduling irregular symbolic computations on computational Grids, and develop more sophisticated parallel skeletons. Finally, we will provide wider access to high performance symbolic computation by offering **SymGrid-Par** as a grid service.

References

1. GridMathematica2, `http://www.wolfram.com/products/gridmathematica/`.
2. *High performance computations in group representation theory.* Preprint, Institut fur Experimentelle Mathematik, Univerisitat GH Essen,, 1998.
3. GENSS, `http://genss.cs.bath.ac.uk/index.htm`, 2006.
4. Geodise, `http://www.geodise.org/`, 2006.
5. The GpH-Maple Interface, `http://www.risc.uni-linz.ac.at/software/ghc-maple/`, 2006.
6. The OpenMath Format,`http://www.openmath.org/`, 2006.
7. 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.
8. 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.

9. B. Amrheim, O. Gloor, and W. Kuchlin. A case study of multithreaded grobner basis completion. In *In Proc. of ISSAC'96*, pages 95–102. ACM Press, 1996.
10. R. Bundgen, M. Gobel, and W. Kuchlin. Multi-threaded ac term re-writing. In *Proc. PASCO'94*, volume 5, pages 84–93. World Scientific, 1994.
11. K. C. Chan, A. Draz, and E. Kaltofen. A Distributed Approach to Problem Solving in Maple. In *Proc. 5th Maple Summer Workshop and Symp.*, pages 13–21, 1994.
12. B. W. Char and et al. *Maple V Language Reference Manual*. Maple Publishing, Waterloo Canada, 1991.
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. M. Cole. Algorithmic Skeletons. In K. Hammond and G. Michaelson, editors, *Research Directions in Parallel Functional Programming*, chapter 13, pages 289–304. Springer-Verlag, 1999.
15. G. Cooperman. Parallel gap: Mature interactive parallel. *Groups and computation, III (Columbus, OH,1999)*, 2001. de Gruyter, Berlin.
16. 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.
17. 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.
18. The GAP Group. Gap – groups, algorithms, and programming, version 4.2, 2000. St Andrews, http://www.gap-system.org/gap.
19. W. Kuchlin. Parsac-2: A parallel sac-2 based on threads. In *AAECC-8*, volume 508 of *Lecture Notes in Computer Science*, pages 341–353. Springer-Verlag, 1990.
20. R. Martínez and R. Pena. Building an Interface Between Eden and Maple. In *Proc. IFL 2003, Springer-Verlag LNCS 3145*, pages 135–151, 2004.
21. K. Morisse and A. Kemper. The Computer Algebra System MuPAD. *Euromath Bulletin*, 1(2):95–102, 1994.
22. D. Petcu, M. Paprycki, and D. Dubu. Design and Implementation of a Grid Extension of Maple, 2005.
23. L. Roch and G. Villard. Parallel computer algebra. In *ISSAC'97*. Preprint IMAG Grenoble France, 1997.
24. D. Tepeneu and T. Ida. MathGridLink – Connecting Mathematica to the Grid. In *Proc. IMS '04, Banff, Alberta*, 2004.
25. P.W. Trinder, K. Hammond, H.-W. Loidl, and S.L. Peyton Jones. Algorithm + Strategy = Parallelism. *J. Functional Programming*, 8(1):23–60, January 1998.