A Grid-based Parallel Maple

Dana Petcu[1,2], Diana Dubu[1,2], Marcin Paprzycki[3,4]

[1] Computer Science Department, Western University of Timişoara, Romania
[2] Institute e-Austria, Timişoara, Romania,
[3] Computer Science Department, Oklahoma State University, USA,
[4] Computer Science Department, SWPS, Warsaw, Poland
{petcu,ddubu}@info.uvt.ro, marcin.paprzycki@swps.edu.pl

Abstract. Popularity and success of computational grids will depend, among others, on the availability of application software. Maple2g is a grid-oriented extension of Maple. One of its components allows the access of grid services within Maple, while another one use of multiple computational units. The latter component is discussed in this paper. It is based on a master-slave paradigm, and it is implemented using Globus Toolkit GT3, mpiJava and MPICH-G2. Preliminary experiments are reported and discussed. These are proving that a reasonable time reduction of computational-intensive applications written in Maple can be obtained by using multiple kernels running on different grid sites.

1 Introduction

Computer algebra systems (CAS) can be successfully used in prototyping sequential algorithms for symbolical or numerical solution of mathematical problems as well as efficiently utilized as production software in large domain of scientific and engineering applications. It is especially in the latter context, when computationally intensive problems arise. Obviously, such applications of CAS can become less time consuming if an efficient method of utilizing multiple computational units is available. This can be achieved in a number of ways: (a) parallelisation provided "inside" of a CAS or (b) parallelization facilitated by the environment in which multiple copies of the CAS are executing; furthermore, utilization of multiple computational units can occur through (c) computations taking place on a parallel computer, or (d) utilization of distributed resources (i.e. grid based computing). Since coarse grain parallelism has been proved to be efficient in an interpreted computation environment such as the CAS, in this paper we are particularly interested in the case of running multiple copies of the CAS working together within a grid. To be able to facilitate this model of parallelism, a CAS interface to a message-passing library is needed and in our research we have developed such an interface for Maple.

Maple is a popular mathematical software that provides an easy to use interface to solve complicated problems and to visualize the results of computations. Our main reason for choosing Maple is that, despite its robustness and user friendliness, we were not able to locate efforts to link Maple with grids. Second, it is well known that Maple excels other CAS in solving selected classes of

problems like systems of nonlinear equations or inequalities [11]. Furthermore, Maple has already a socket library for communicating over the Internet. Finally, distributed versions of Maple have been recently reported in [6] and [9].

We intend to provide an environment where utilization of multiple computational units is possible within the Maple environment such that the programmer does not have to leave the familiar CAS interface. While several parallel or distributed versions of Maple, mentioned in Section 2, were developed for clusters of computers, our goal is to "port" Maple to the computational grid. Success of our project will allow multiple grid users with Maple installed on their computers, to pool their resources and form a distributed Maple environment.

We have therefore proceeded to develop Maple2g: the grid-wrapper for Maple. It consists of two parts: one which is CAS-dependent and another, which is grid-dependent. In this way, any change in the CAS or the grid needs to be reflected only in one part of the proposed system. The CAS-dependent part is relatively simple and can easily be ported to support another CAS or a legacy code. Maple2g should therefore be portable to any new commercial version of Maple, including the Socket package. The system only relies on basic interfaces to the Maple kernel. More details about Maple2g architecture are given in Section 3. We continue by describing, in Section 4, the proposed approach to distributed computing. Finally, results of our experiments are presented in Section 5, while conclusions and future improvements are enumerated in Section 6.

2 Parallel and distributed versions of Maple

With the increasing popularity of parallel and distributed computing, researchers have been attempting at building support for parallel computing into Maple. We are aware of the following attempts to supplement Maple with parallel and/or distributed computation features.

||Maple|| is a portable system for parallel symbolic computations built as an interface between the parallel programming language Strand and Maple [10]. Sugarbush combines the parallelism of C/Linda with Maple [2]. Maple was ported also to the Intel Paragon architecture [1]. Five message passing primitive were added to the kernel and used to implement a master-slave relationship amongst the nodes. The manager could spawn several workers and asynchronously await the results. Finally, FoxBox provides an MPI-compliant distribution mechanism allowing parallel and distributed execution of FoxBox programs; it has a client/server style interface to Maple [3].

All these attempts took place in the 1990th and are all but forgotten. In recent years we observe a renewed interest parallel/distributed Maple.

Distributed Maple is a portable system for writing parallel programs in Maple, which allows to create concurrent tasks and have them executed by Maple kernels running on separate networked computers. A configurable program written in Java starts and connects external computation kernels on various machines and schedules concurrent tasks for execution on them. A small Maple package implements an interface to the scheduler and provides a high level parallel programming model for Maple [9].

Parallel Virtual Maple (PVMaple) was developed to allow several independent Maple kernels on various machines connected by a network to cooperate in solving a problem. This is achieved by wrapping Maple into an external system which takes care of the parallel execution of tasks: a special binary is responsible for the message exchanges between Maple processes, coordinates the interaction between Maple kernels via PVM daemons, and schedules tasks among nodes [6]. A Maple library implements a set of parallel programming commands in Maple making the connections with the command messenger. The design principles are very similar to those of the Distributed Maple.

The above mentioned parallel or distributed Maple projects make use of message-passing for interprocessor communication and provide message-passing interfaces to the user. Commands, like *send* and *receive*, are available so that a user can write a parallel Maple program using the message-passing programming model. These commands are implemented either in user written subroutines callable by Maple or in script files written in Maple's native programming language. They utilize low level message-passing routines from the standard MPI/PVM libraries, simple communication functions, or file synchronization functions (in the case of a communication via a shared file system). Actually, existing parallel/distributed versions Maple can be regarded as a *message-passing extension of Maple*. A recent more comprehensive description of the available parallel and distributed versions of Maple can be found in [9].

None of the attempts at adding parallel/distributed features to Maple, that we were able to located tried to introduce Maple to the grids. It is the latter idea that became focus of our work. Our experiments with PVMaple showed sufficient efficiency in solving large problems to follow this design paths in Maple2g development. As will be seen below, the later has similar facilities with PVMaple.

3 Maple2g architecture

Rewriting a CAS kernel in order to supplement its functionality with grid capabilities can be a complicated and high-cost solution. Wrapping the existing CAS kernel in an interface between the grid, the user and the CAS can be done relatively easily as an added functionality to the CAS. In addition, it can also be adapted on-the-fly when new versions of the CAS in question become available. In this way Maple2g is a prototype grid-enabling wrapper for Maple.

Maple2g allows the connection between Maple and computational grids based on the Globus Toolkit. The prototype consists of two parts. A CAS-dependent part (*m2g*) is the Maple library of functions allowing the Maple user to interact with the grid or cluster middleware. A grid-dependent part (*MGProxy*) is the middleware, a package of Java classes, acting as interface between m2g and the grid environment. The m2g functions are implemented in the Maple language, and they call MGProxy which accesses the Java CoG API [5]. A preliminary description of fundamental Maple2g concepts is present in [7].

Maple2g has three operating modes: *user mode*, for external grid-service access, *server mode*, for exposing Maple facilities as grid services, and *parallel mode* for parallel computations in Maple using the grid.

In the current version of Maple2g we have implemented a minimal set of functions allowing the access to the grid services:

`m2g_connect()`: connection via Java COG to the grid;
`m2g_getservice(`c, l`)`: search for a service c and retrieve its location l;
`m2g_jobsubmit(`t, c`)`: job submission on the grid based of the command c;
`m2g_results(`t`)`: retrieve the results of the submitted job labeled t.

More details about the implementation of the grid services access procedures from Maple illustrated by several examples can be found in [8]. Let us now proceed to present some details of utilizing Maple2g in parallel on the grid.

4 Coupling Maple kernels over grid - Maple2g approach

The computational power of a CAS can be augmented by using several other CAS kernels (the same or different CASs) when the problem to be solved can be split between these kernels or a distributed-memory parallel method is used in order to solve it. The usage of a standard message-passing interface for inter-kernel communication allows the portability of the parallel version of a CAS in particular an easy deployment on clusters and grids (Figure 1).

The two extreme approaches to design the interaction with the message-passing interface are minimal, respectively full, access to the functions of the message-passing interface. In the first case the set of functions is restricted to those allowing to send commands and receive results from the remote kernels. In the second case it is possible to enhance the CAS with parallel or distributed computing facilities, allowing the access of the CAS to other parallel codes than the ones written in the CAS language (the message-passing interface can be used as interpreter between parallel codes written in different languages). The first approach has been followed in our Maple2g prototype.

Parallel codes using MPI as the message-passing interface can be easily ported to grid environments due to the existence of the MPICH-G2 version which runs on top of the Globus Toolkit. On other hand, the latest Globus Toolkit GT3 is built in Java, and the Java clients are easier to write. This being the case, we selected mpiJava as the message-passing interface between Maple kernels.

In Maple2g a small number of commands have been implemented and made available to the user, for sending commands to other Maple kernels and for receiving their results (Table 1).

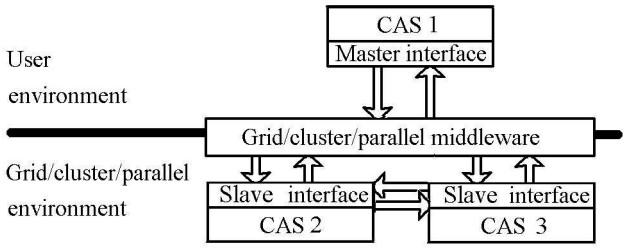

Fig. 1. Coupling CAS kernels over the grid using a master-slave approach

Table 1. Maple2g functions/constants for remote process launch/communications

Function/const.	Description
m2g_maple(p)	Starts p processes MGProxy in parallel modes
m2g_send(d, t, c)	Send at the destination d a message labeled t containing the command c; d, t are numbers, c, a string; when d is "all", c is send to all kernels
m2g_recv(s, t)	Receive from the source s a message containing the results from the a previous command labeled t; when s is 'all', a list is returned with the results from all kernels which have executed the command t
m2g_rank	MGProxy rank in the MPI World, can be used in a command
m2g_size	Number of MGProxy processes, can be used in a command

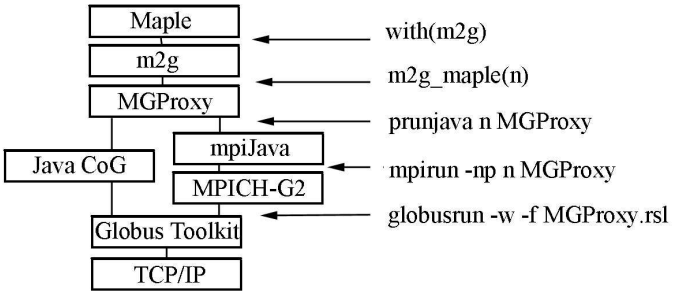

Fig. 2. From a m2g command to a grid request

Maple2g facilities are similar to those introduced in the PVMaple [6]. The user's Maple interface is seen as the master process, while the other Maple kernels are working in a slave mode. Command sending is possible not only from the user's Maple interface, but also from one kernel to another (i.e. a user command can contain inside a send/receive command between slaves).

Figure 2 shows how the m2g_maple command is translated in a grid request.

MGProxy is activated from user's Maple interface with several other MGProxy copies by m2g_maple command. The copy with the rank 0 enters in user mode and normally runs in the user environment, while the others enter in server mode. Communication between different MGProxy copies is done via mpiJava.

5 Test results

We have tested the feasibility of Maple2g approach to development of distributed Maple applications on a small grid based on 6 Linux computers from two locations: 4 PCs located in Timisoara, Romania; each with a 1.5 GHz P4 processor and 256 Mb of memory, connected via a Myrinet switch at full 2Gb/s and 2 computers located at the RISC Institute in Linz[5]; one with a P4 processor running at 2.4 GHz and 512 Mb, and a portable PC with a 1.2 GHz PIII processor and 512 Mb, connected through a standard (relatively slow) Internet connection.

Note that the one of the possible goals of using the Maple kernels on the grid is to reduce the computation time (it is **not** to obtain an optimal runtime, like

[5] In the frame of the IeAT project supported by Austrian Ministries BMBWK project no. GZ 45.527/1-VI/B/7a/02, BMWA project GZ no. 98.244/1-I/18/02

```
>with(m2g); m2g_MGProxy_start();
[m2g_connect,m2g_getservice,m2g_jobstop,m2g_jobsubmit,m2g_maple,m2g_rank,
m2g_recv,m2g_results,m2g_send,m2g_size,m2g_MGProxy_end,m2g_MGProxy_start]
       Grid connection established
>p:=4: a:=1: b:=2000: m2g_maple(n);
       Connect kernel 1: successful
       Connect kernel 2: successful
       Connect kernel 3: successful
       Connect kernel 4: successful
>m2g_send("all",1,cat("s:=NULL:a:=",a,":b:=",b,": for i from a+m2g_rank",
 " to b by m2g_size do if isprime(i*2^i-1) then s:=s,i fi od: s;"):
>m2g_recv("all",1);
       [[81,249],[2,6,30,362,462,822],[3,75,115,123,751],[384,512]]
>m2g_MGProxy_end();
       Grid connection closed
```

Fig. 3. Maple2g code and its results searching all the Woodall primes in $[a, b]$

on a cluster or a parallel computer. When a grid user executes a parallel Maple program, other computers typically become available as "slaves". Taking into account the possible relative slowness of the Internet (and unpredictable connection latency), it would be costly to pass data frequently among the computational nodes. This being the case, the best possible efficiency for embarrassingly parallel problems; for example, when each slave node receives a work package, performs the required computation, and sends the results back to the master. In what follows we present two examples of such computations, which therefore can be treated as the "best case" scenarios.

There are several codes available on Internet to solve in parallel open problems like finding prime numbers of specific form [4]. For example, currently the Woodall numbers, the primes of the form $i2^i - 1$, are searched in the interval $[10^5, 10^6]$. Figure 3 presents the Maple2g code and its results searching the Woodall primes in a given interval $[a, b]$ using $p = 4$ Maple computational units.

A second example involves graphical representation of results of a computation. Given a polynomial equation, we count the number of Newton iterations necessary to achieve a solution with a predescribed precision and starting from a specific value on the complex plane. If we compute these numbers for the points of a rectangular grid in a complex plane, and then we interpret them as colors, we may obtain a picture similar to that from Figure 4. The same figure displays the Maple2g code in the case of using 4 kernels; vertical slices of the grid are equally distributed among these kernels.

We have run our experiments on two basic combinations of available computers. First, using 4 machines clustered in Timişoara and, second, using 2 machines in Timişoara and 2 in Linz. We have experimented with a number of possible approaches to the solution of the two problems, where both problems come in two different sizes representing a "small" and a "large" problem. Table 2 summarizes the results and the notations used there refers to the following case studies:

`Sequential:` the Woodall prime list and the plot were constructed without any splitting technique and the results come form one of the PC's in Timişoara.

```
>with(m2g): m2g_MGProxy_start(); no_procs:=4;
>m2g_maple(no_procs): d:="all";
>m2g_send(d,1,"f:=x->x^7+x^6+5*x^5+3*x^4+87*x^3
            +231*x^2+83*x+195:"):
>m2g_send(d,2,"newton:=proc(x,y) local z,dif,m;
       dif:=1; z:=evalf(x+y*I);
       for m to 30 while abs(dif)>0.1*10^(-8) do
       dif:=f(z)/D(f)(z); z:=z-dif od; m end:");
>m2g_send(d,3,"plot3d(0,-5+10*m2g_rank/m2g_size..
         -5+10*(m2g_rank+1)/m2g_size, -5..5, grid=[160/m2g_size,160],
         style=patchnogrid,orientation=[90,0],color=`newton`);"):
>plots[display3d](m2g_recv(d,3)); m2g_MGProxy_end();
```

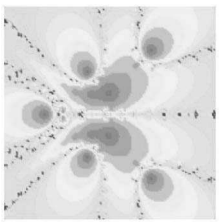

Fig. 4. Maple2g code and the graphical result in measuring the levels of Newton iterations to solve in the complex plane a polynomial equation of degree seven

`Ideal:` the maximum possible reduction of the time using p processors;

`Cycle:` the Woodall prime list or the plot were constructed in a sequential manner, but in a cycle with p steps. The time per step is variable. We registered the maximum of the time value of each step (on a cluster's PC).

`MPI-cluster:` The codes from the Figs. 3 and 4 are used on p processors of the cluster; here mpiJava is installed over MPICH version; Globus is not used.

`G2-cluster:` Same codes were used on p processors of the cluster in Timişoara, here mpiJava is installed over MPICH-G2 using Globus Toolkit 3.0.

`G2-net:` Same codes were used on p different processors running the mpiJava based on MPICH-G2: in the case of $p = 2$, one PC in Timişoara and the faster machine in Linz are used; in the case of $p = 4$, two PC's in Timişoara and two computers in Linz are used. The parallel efficiency is computed.

Overall, it can be said that a reasonable parallel efficiency for the larger problem has been achieved: for the Woodall primes: 73% for 2 processors and 44% for 4 processors; for Newton iteration visualization: 75% for 2 processors and 40% for 4 processors. As expected, efficiency is improving as the problem size is increasing.

At the same time the results are somewhat disturbing when one considers the current state of grid computing. On the local cluster, the results based on

Table 2. Time results

Problem	Woodall primes		Newton iterations	
p Implementation	$[a,b]$=[1,2000]	$[a,b]$=[1,4000]	grid=160 × 160	grid=300 × 300
1 Sequential	236 s	3190 s	208 s	1804 s
2 Ideal	118 s	1595 s	104 s	902 s
Cycle	122 s	1643 s	105 s	911 s
MPI-cluster	135 s	1725 s	123 s	1020 s
G2-cluster	153 s	1846 s	138 s	1071 s
G2-net	185 s	2197 s	160 s	1199 s
4 Ideal	59 s	797 s	52 s	451 s
Cycle	65 s	885 s	55 s	473 s
MPI-cluster	79 s	1027 s	73 s	654 s
G2-cluster	107 s	1263 s	94 s	784 s
G2-net	160 s	1831 s	138 s	1129 s

mpiJava and MPICH are substantially better than these obtained when the mpiJava and MPICH-G2 are used. This indicates a considerable inefficiency in the MPICH-G2 package. Furthermore, our results indicate that currently, realistic application of grids over the Internet makes sense only for very large and easy to parallelize problems (like seti@home). For instance, when machines residing at two sites were connected then the efficiency dropped by about 18%. Obviously, in this case this is not the problem with the grid tools, but with the Internet itself. However, since the grid is hailed as the future computational infrastructure, and since our two problems represented the best case scenario, it should be clear to everyone that, unfortunately, we are far away from the ultimate goal of the grid paradigm.

6 Conclusions and future developments

At this stage, the proposed extension of Maple exists as a demonstrator system. Maple2g preserves the regular Maple instruction set and only add several new instructions. Further work is necessary to make it a more comprehensive package and to compare it with similar tools build for clusters. In this paper we have shown that utilizing Maple2g allows developing grid-based parallel applications. Our initial test have also indicated satisfactory efficiency of Maple2g, especially when native MPI tools are used (instead of their Globus based conuterparts). In the near future we plan intensive tests on grids on a large domain of problems to help guide further development of the system. Among others, the master-slave relationship between nodes will be extended to allow slaves to become masters themselves and thus facilitate the development of hierarchical grid applications.

References

1. Bernardin, L.: Maple on a massively parallel, distributed memory machine. In Procs. 2nd Int. Symp. on Parallel Symbolic Computation,Hawaii (1997), 217-222.
2. Char B. W.: Progress report on a system for general-purpose parallel symbolic algebraic computation. In ISSAC '90, ACM Press, New York (1990).
3. Diaz A., Kartofen E.: FoxBox: a system for manipulating symbolic objects in black box representation. In ISSAC '98, ACM Press, New York (1998).
4. Internet-based Distributed Computing Projects, www.aspenleaf.com/distributed/.
5. Java CoG Kit, http://www-unix.globus.org/cog/java/.
6. Petcu D.: PVMaple – a distributed approach to cooperative work of Maple processes. In LNCS 1908, eds. J.Dongarra et al., Springer (2000), 216–224
7. Petcu D., Dubu D., Paprzycki M.: Towards a grid-aware computer algebra system, In LNCS 3036, eds. M.Bubak, J.Dongarra, Springer (2004), 490–494.
8. Petcu D., Dubu D., Paprzycki, M.: Extending Maple to the grid: design and implementation. Procs. ISPDC'2004, Cork, July 5-7, 2004, IEEE series, in print.
9. Schreiner W.,Mittermaier C.,Bosa K.: Distributed Maple–parallel computer algebra in networked environments.J.Symb.Comp.35(3),Academic Pr.(2003),305–347.
10. Siegl K.: Parallelizing algorithms for symbolic computation using ||Maple||. In Procs. 4th ACM SIGPLAN Symp. ACM Press, San Diego (1993), 179–186.
11. Wester M.: A critique of the mathematical abilities of CA systems. In CASs - A Practical Guide, ed. M.Wester, J.Wiley (1999), math.unm.edu/~wester/cas_review