An Extension of Maple for Grid and Cluster Computing

Petcu Dana

Dubu Diana

Western University of Timişoara and Institute e-Austria in Timişoara

ROMANIA

Abstract: Computer Algebra Systems, like Maple, are useful tools for mathematicians and engineers. Extending their facilities with external specialized codes is a continuous challenge. In order to augment Maple's facilities with dedicated remote services or using remote computational power, we have design a specific tool. This tool links Maple's interface to grid services or different Maple kernels thus forming a virtual parallel computing environment. Behind it are Globus, the standard for grid computing, and mpiJava, a package for message passing programming on parallel or cluster architecture. In this paper we give a shortly description on the extension design principles and its functionality.

Keywords: grid computing, parallel computing, computer algebra systems**.**

Dana Petcu is professor at Computer Science Department of the Faculty of Mathematics and Computer Science from Western University of Timişoara and director of the research institute in computer science, Institute e-Austria in Timişoara. Her current research interests include parallel, distributed and grid computing, numerical analysis, computer graphics. She is the author of more than 10 books and 100 papers published in international journals or conference proceedings.

Diana Dubu has finish the master degree in Distributed systems and artificial intelligence at Computer Science Department of the Faculty of Mathematics and Computer Science from Western University of Timişoara and is young researcher at Institute e-Austria Timişoara. Her current research interests include grid computing and automated proving.

1. Introduction

Grid computing is emerging as a next generation computing platform for solving large scale problems through sharing of geographically distributed resources [1]. It is unrealistic to expect the transition of researchers towards grid computing to be anything but halting and slow if it means abandoning the scientific computing environments, like computer algebra systems (CAS), which are now rightfully viewed as a major source of their productivity. Moreover, there are many pieces of functionality which are implemented in one CAS but not others, as stand-alone programs, or which run best on special hardware such as parallel machine. Whichever is the case, it is desirable to be able to augment each CAS with functionality from external systems [11]. A CAS extension to the grid meets not only the need of interconnecting different solvers, but also opens a way for web-based interaction with large-scale computational systems.

There are several running projects aiming to link CAS with grids. NetSolve [2] is an example of a grid-based server that supports Matlab and Mathematica as native clients for grid computing; it automates the process of looking for computational resources on a network, choosing the best one available, solving a problem, and returning the answer to the user. MathGridLink [13] permits the full access to any available web service deployed on the grid, regardless of the language the service is written in, within Mathematica; with MathGridLink it becomes possible not only to integrate any existing grid web service in Mathematica's framework, but also to write and deploy new web services entirely from Mathematica, such that the Mathematica programmer does not need to have detailed knowledge about the grid. The Geodise toolkit [4] is a suite of tools for grid-enabled codes within the Matlab environment; these grid services are presented to the design engineer as Matlab functions that conform to the usual syntax of Matlab, enabling the programmer to access Java classes, to create objects, and to call methods of these objects within the Matlab environment.

Many scientific and engineering problems are characterized by a considerable run-time expenditure on one side and a medium-grained logical problem structure on the other side. Often such problems cannot be solved in a single CAS, but they can be parallelized effectively even on networked computers. For such problems we do not necessarily need a parallel CAS. Instead we need some methods to integrate the CAS and other autonomous tools into a parallel virtual system. Several parallel virtual or distributed systems were constructed on top of three frequently used CAS, Maple, Matlab and Mathematica. Distributed Mathematica [12] is a public domain system allowing to create concurrent tasks and have them executed by Mathematica kernels running on different machines on a cluster. Another extension for Mathematica, gridMathematica [14], allows the parallel distribution of Mathematica tasks among

different kernels in a distributed environment, built on a PVM-like architecture; it can be used on dedicated multiprocessor machines and on clusters. In [3] several tens of parallel Matlab projects have been identified utilizing different approaches: compile Matlab script into parallel native code, provide a parallel backend to Matlab by using Matlab as a graphical front-end, or coordinate multiple Matlab processes to work in parallel.

While there are several parallel and distributed versions of Maple (see [10] for a review of them, including the one proposed earlier by one of the current paper authors), or network-enabled Maple servers (see [6]), we are not aware of any current project running and reported in the literature with the aim to couple Maple to a computational grid. To fill this gap we have designed an extension of Maple allowing the access to grid services from Maple and the access of Maple as grid service. Moreover, the extension allows parallel computations using different Maple kernels running on clusters of workstations or on several nodes of a computational grid.

The paper is organized as follows. The next section enumerates the design approach. The third section gives details about the functionality of the grid extension, while the fourth section explains the background of the parallel extension. The last section is dedicated to future developments and conclusions. Further details are presented in [9].

2. Design Issues

Instead of designing a whole new system for interconnecting several computation services, we can rely on some already available and viable solutions. We review here some of them.

A client proxy is a process that resides on the client host and acts on behalf of the client to handle all interactions with a specific kind of grid backend. Each different backend has its own characteristics and requires its own proxy to shield the client from the details of that interaction. For example, to enable NetSolve to interact with the Globus system a proxy was built for the NetSolve client that knows how to interact with and make use of Globus resources [2]. By using proxies that abstract away the details of the system communication, the client interface can stay consistent while the grid systems it can utilize can change and grow.

To transform a component system's interface type and syntax in order for it to become part of a conglomerate system, like a computational grid, a piece of software known as a wrapper must be implemented. A wrapper [11] presents the desired interface to the operating environment and translate external interactions to and from the native interface and syntax of the component system. The construction of wrappers is probably the most difficult part of building conglomerate systems, especially when the native interface of a system is a user interface, with all the attendant ambiguities of interpretation of output. Automatic wrapper generators for legacy codes that can operate at varying degrees of granularity, and can wrap the entire code or sub-routines within codes automatically, are still not available.

The steps required to add functionality from an external system to a CAS can be conceptualized as follows:

Step 1: install and maintain the component which performs the computation;
Step 2: write the wrapper for the component with a command-line interface and formal syntax;
Step 3: devise a scheme whereby the wrapper can be invoked from within the CAS.

The difficulty associated with the first step consist in that the user of a CAS usually does not want the inconvenience of installing and maintaining an external system. The second step is often the most challenging because it requires disambiguating the user output of the original system into formal output syntax of the wrapper. The main problem with the third step is the fact that most CASs do not support a programmatic interface to external programs and must therefore perform all interaction through the command shell and input/output files. Fortunately, Maple has a sockets library for communicating over the Internet, and a library for parsing XML. These ingredients suffice to make Maple a client for a computation service available on the Internet.

To achieve a single generic mechanism which could be used for all computation requests with little or no extra software needing to be loaded into the CAS to interface with each new online service, a problem is to determine how CASs will phrase requests for computation and how they will interpret the results. Therefore it is necessary to establish a standard for request-response exchanges. Part of this is a standard for the representation of the mathematical objects exchanged. MathML [7], also used by Maple, is well advanced on the path to solving this problem.

Another problem is to locate the service providing a particular computation without knowing its location beforehand. This requires a mechanism for discovery. Globus Toolkit [5] can be used to solve this problem.

We propose an Maple extension, Maple2g (Maple-to-grid). Maple2g is a prototype of an wrapper for grid-enabled version of Maple. It has three components:

MGProxy: a Java interface for Maple with the grid/cluster environment;

M2g library: a Maple library of functions allowing the user to interact with the grid/cluster environment;

Thin client: prototype web interface to access the Maple web/grid/cluster services.

The user exposed functions are implemented in the Maple language, and they call Java classes which access the Java CoG API [5] for grid services, respectively mpiJava API for parallel computing.

MGProxy has three operating modes:

user mode: activating from Maple, receives the user command from the user's Maple interface via a socket interface, contacts (at user request) the grid/cluster services, including other MGProxy in server/parallel modes, queries the user requests to the contacted services, and at user requests sends to the main Maple interface the queries results.

server mode: activates a Maple twin process (which enters in a infinite cycle of interpreting commands coming via a socket interface), acts as server waiting for external calls, interprets the requests, sends the authenticate requests to the Maple twin process, enters in receive mode of the Maple results, and sends them back to the user.

parallel mode: it is activated from user's Maple interface with several other MGProxy copies; the one with the rank 0 enters in user mode and normally runs in the user environment, while the others enter in server mode; the communication between different MGProxy copies is done via the message passing interface.

MGProxy communicates with Maple via sockets. The Maple commands are circulating in the systems as strings, and the Maple results are presented in MathML format.

3. Accessing Grid Services from Maple

One approach in grid-enabling the computer algebra system is to make grid services available to the user. The coupling with the services exposed within the grid has to be performed in a transparent way for the user, such that the service methods call should be only in the CAS native language syntax. The suite of Maple functions, incorporated in the package m2g, allows the programmatic access to Globus Grid enabled resources. The m2g package translates internally the functions from the syntax familiar to the Maple user into commands, allowing the initiation and further communication with the MGProxy middleware. MGProxy, acting as an intermediary between Maple and the grid, is written in Java since this platform is portable and libraries specialized for the grid computing have been already implemented for Java in the Globus Toolkit. Globus is today's "the facto" tool for the grid architecture. Java CoG (i.e. Commodity Grid) Kit provided by Globus integrates the software for grid computing developed by Globus and the Java commodity framework, thus facilitating the development and deployment of grid services, while also permitting the use of web services as parts of the grids.

The procedure is as follows:

Start MGProxy: User commands in Maple syntax, parsed within the m2g package, initiate the communication with MGProxy intermediary. This is performed via the commands for external code invocation (including Java) which are already available in the latest versions of Maple.

Issue commands: The user invokes the remote services by issuing commands in RSL syntax

Job Submission: MGProxy activates GridJob (described bellow), which is a Java class encapsulating GRAM job that deals with job submission over the grids.

Prompt for results: Results can be requested either during the communication or even after closing the connection with the grids.

The suite of functions incorporated in m2g Maple package for grid services is the following one:

m2g_proxyinit():	Starts Globus grid_proxy_init and returns the text provided by it.
m2g_getservice(c,l):	Search for a service c and give a link to it, retrieve its location l.
m2g_jobsubmit(t,c):	Allows a job submission in the grid environment labeled with the number t: the command from the string c is send to the MGProxy which treats it as a grid-service request.
m2g_status(t):	Queries the status of the submitted job labeled t.
m2g_results(t):	Retrieve the results of the submitted job labeled t.

A short example of using these functions is shown in Figure 1.

```
> with(m2g): m2g_MGProxy(); m2g_proxyinit();
> m2g_getservice("factor",`service_access`)
> m2g_jobsubmit(1,cat(service_access,"factor ","234567890"));
> m2g_results(1);
```

Figure 1: Accessing in Maple an External Factorization Code, Available as Grid Service

The GridJob class is responsible for the job submission. Requests received from the Maple's user interface in RSL syntax are send over the grids to remote resources. The underlying framework used is Java CoG. The connection is established with the remote server, referred to as the 'gatekeeper', which is responsible for the execution of the job (i.e. a binary executable or command to be run remotely). Both the host and the gatekeeper must comply with the authentication requirements. The Grid Security Infrastructure (GSI) is used for enabling secure authentication and communication over an open network. Globus uses GASS for porting and running the applications requiring I/O files to the Grid environment. Therefore, GridJob starts a GASS server and submits all GRAM job requests to this server. The request is formatted accordingly in the RSL format. The output of the processed job is returned through the MGProxy intermediary to Maple in the user's interface. The communication flow is depicted in Figure 2.

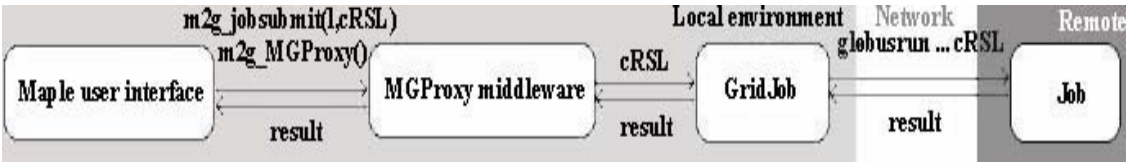

Figure 2: Communication Flow when it is Access a Grid Service (Depicted as Job) from Maple}

4. Cooperative Multiple Maple Kernels

The CAS users are often frustrated by the time to rerun a program for different conditions, parameters or initial guesses. Such user's problems might be solved by a system that makes it convenient to spawn CAS processes on multiple processes of a parallel computer or a cluster. In many cases the needs for communication between the processors are rather small compared with the necessary computations. The computational power given by a CAS and its running environment can be augmented by using several other CAS kernels (the same or different CASs) when the problem to be solved is properly split between this kernels and/or a distributed-memory parallel method is used to solve it. Using a standard message-passing interface for inter-kernel communication allows the portability of the parallel version of a CAS towards distributed environments, in particular towards clusters.

The two extreme ways to design the interaction with the message-passing interface are fully versus minimal access to the functions of a message-passing interface. In the first case it is possible to enhance the CAS with parallel or distributed computing facilities, allowing the access into a 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). In the second case the set of functions is restricted to those allowing to send commands and receive results from the remote kernels. Distributed Maple [12] uses Java socket library to allow the execution of concurrent tasks of Maple kernels running on different machines of a network, while PVMaple [10] uses PVM C library.

Parallel codes using MPICH as message-passing interface can be easily ported to grid environments due to the existence of a MPICH-G version on top of Globus Toolkit. On other hand, the latest Globus Toolkit is build on Java, and the Java clients are easier to be written for a computational grid. In this condition, we selected mpiJava [8] as message-passing interface between Maple kernels.

Only a small number of commands are available to the user, mainly concerning sending commands to other Maple kernels and receiving their results:

mg_run(p):	Starts p processes MGProxy in parallel modes.
mg_send(d,t,c):	Send at the destination kernel numbered d a message labeled t containing the Maple command c; d and t are numbers, c is a string; when 'all' is used in destination field, the command is send to all Maple kernels accept the current one.
mg_recv(s,t):	Receive from source kernel s a message labeled t containing the results from a previous command labeled t; s and t are numbers; when s is 'all', a list is returned with the results from all kernels which have been executed the command labeled **t.**
mg_rank:	MGProxy rank in the MPI World, can be used in a Maple command.
mg_size:	The number of MGProxy processes, can be used in a Maple command.

These facilities are similar to those introduced by PaViS [10]. The communication flow in the case of two kernels is depicted by Figure 3.

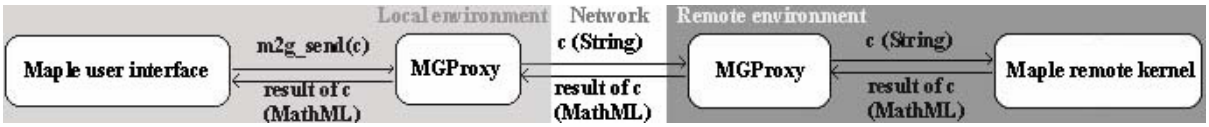

Figure 3: Communication Flow with Remote Maple Kernel Which Execute a Maple Command (namely c)

Figure 4 shows an example of using Maple2g in order to improve the time response of Maple. In this case the sequential time necessary to factorize the list of integers increases linearly with the problem dimension (list length). The sequential execution time reported by Maple is around 100 s for 4000 integers. Splitting the list of integers and the factorization requests between several Maple kernels running on remote processors (of a cluster of PCs with Intel PIV processors at 1.5 MHz, connected by a Myrinet network at 2Gbs), we can obtain a shorter response time. Indeed, for the given case, a speed-up factor of 3.2 was registered using 4 kernels of Maple. We expect to obtain higher speed-up values by increasing the problem dimension since the communication time increases slower with the number of integers than the computation time.

```
> with(m2g):
 > intfac=proc(dim,big,p) local n,r,i;
            r:=rand(big); n:=[];  for i to dim do n:=[op(n),r()]; od;  nfactor(n,p); end:
 > nfactor:=proc(n,p) local ,mes;
              mes=cat(`readlib(ifactors): n:=`,convert(n,string),`:s:=[]: for i to nops(n) do`,
              `if(i mod`,convert(p,string),`)=m2g_rank-1 then s:=[op(s),ifactors(n[i])]: fi:`,
              `od: s;`); m2g_send(all,10,mes); m2g_recv(all,10) end:
 > m2g_MGProxy(); m2g_run(4): intfac(4000,2^32,nops(m2g_size)-1);
```

Figure 4: Code for Distributed Integer Factorization Within Maple2g Sessions: 4000 Integers are Randomly Generated (Intfac Function), Distributed in a Round-robin Fashion to 4 Maple Kernels, and Factorized (by n Factor Procedure), the Final List of Factors Being Presented in the User's Maple Interface

5. Future Work

The proposed wrapper for grid and parallel computing within Maple is in a prototype phase. Several tests are planned to be performed on larger grid and cluster environment. The problem database of these tests must be further developed to provide a real image on the system performance. Further facilities must be also implemented to align the proposed package to the number and quality of those facilities provided by Netsolve, Geodise or MathGridLink.

REFERENCES

1. ***Globus Toolkit**, http://www.globus.org
2. ***MapleNet**, http://www.maplesoft.com/maplenet/
3. ***MathML**, The W3C's Math Homepage, http://www.w3.org/Math/
4. ***mpiJava, http://www.npac.syr.edu/projects/pcrc/HPJava/mpiJava.html
5. ***Wolfram Research, **gridMathematica**, http://www.wolfram.com.
6. BUYYA R., BAKER, M., Preface, **Grid Computing** 2000, LNCS 1971, 2000.
7. CASANOVA H. AND DONGARRA J., NetSolve: **A Network Server for Solving Computational Science Problems.** Internat. Journal of Supercomputer Applications and High Performance Computing**,** 11(3):212--223, 1997.
8. CHOY R., EDELMAN A.: **Matlab*P 2.0: a unified parallel MATLAB,** In Procs. 2nd Singapore-MIT Alliance Symp. 2003, in print.
9. ERES M. H., POUND G.E., JIAO Z., WASON J.L., XU F., KEANE A.J., AND COX J.S., **Implementation of a grid-enabled problem solving environment in Matlab.** In Procs. WCPSE03 2003, in print, http://iccs03.cs.uwa.edu.au/program/WorkshopsICCS2003Page2.html, www.geodise.org
10. PETCU D., DUBU D., M. PAPRZYCKI, **Towards a Grid-aware Computer Algebra System,** In LNCS 3036, eds. M.Bubak, J.Dongarra, Springer, 2004, 490-494.
11. PETCU D. AND GHEORGHIU D., **PAVIS: a parallel virtual environment for solving large mathematical problems.** In Parallel Computing. Advances and Current Issues, eds. G.R. Joubert, A. Murli, F.J. Peters, M. Vanneschi, Imperial College Press, 2002, 490-497.
12. SOLOMON A.: **Distributed computing for conglomerate mathematical systems.** In Integration of Algebra and Geometry Software Systems, eds. M. Joswig, N. Takayama, http://www.illywhacker.net/ papers/webarch.ps
13. SCHREINER W., **Developing a distributed system for algebraic geometry**, in Procs. {EuroCM-Par'99}, ed. B. Topping, Civil-Comp. Press, Edinburgh, 1999, 137-146.
14. TEPENEU D. AND IDA T., MathGridLink - **A bridge between Mathematica and the Grid**. In Procs. JSSST03, 2003, in print.