CHAPTER 27

Symbolic Computations on Grids

Dana Petcu,[1] Dorin Ţepeneu,[2] Marcin Paprzycki,[3] Tetsuo Ida[2]

[1]*Institute e-Austria Timişoara, and University of Timişoara, Romania*
[2]*University of Tsukuba, Japan*
[3]*Oklahoma State University, USA*

CONTENTS

1. INTRODUCTION

There exist two basic approaches to computational solution of mathematical problems: numerical and symbolic. For a long time, the numerical approach had an advantage of being capable of solving a substantially larger set of problems. Recently the symbolic approach gained more recognition as a viable tool for solving large-scale engineering problems. Symbolic solution of mathematical problems involves manipulations of symbolic objects, like logical or algebraic formulae, rules or programs. Unlike the numerical approach, one of the main goals of the symbolic approach is exactness. Typically, the final answer is either a rational number, or a formula that represents the answer. In this way symbolic calculations can be used (a) to find an exact solution to a given problem, (b) to study the problem when only partial information is available and the formula obtained through symbolic calculations represents simplification of the mathematical model through application of existing knowledge about the problem in question, and (c) in simulations, where the mathematical formula is parameterized and used to study a range of solutions depending on the choice of parameters.

Developments in symbolic computing are lagging relative to numerical computing, mainly due to the inadequacy of available computational resources: most importantly computer memory, but also processor power. Continuous growth in the capabilities of computer hardware led to an increasing interest in symbolic calculations and resulted, among others things, in development of sophisticated Computer Algebra Systems (CASs). These systems allow

Engineering the GRID
Edited by B. Di Martino, J. Dongarra, A. Hoisie, L. T. Yang, and H. Zima
Pages (1–17)
ISBN: 1-58883-038-1

users to "directly" study computational problems on the basis of their mathematical formulations and to focus on the problems themselves instead of spending time transforming the problems into forms that are numerically solvable. As an effect, symbolic computations are being applied in a number of diverse disciplines such as, among others, pure and applied mathematics, physics, engineering and economics [1]. Symbolic computation is also becoming a basis for advanced applications in many areas of computer science, such as computer aided design or software development, VLSI design, geometric modeling and reasoning, robot programming etc. Finally, symbolic methods have also become popular in life sciences, in particular in studying human genome.

While the major purpose of the CAS is to manipulate formulas symbolically, many systems have substantially extended their capabilities. Nowadays, robust CASs offer other functionalities like graphics and simulations allowing a more comprehensive approach to problem solving. Furthermore, modern CASs are capable of solving very large problems. While, typically, CAS systems are utilized in an interactive mode, to solve large problems they can be also used in a "batch" mode and programmed using languages that are very close to common mathematical notation. Lists of existing CASs can be found in Refs. [2–4] and their facilities were compared in Refs. [5–7].

As CASs become capable of solving large problems, they follow the course of development that has already been taken by numerical software: from sequential computers to parallel machines to distributed computing and finally to the grid. It is particularly the grid that has high potential as a discovery accelerator. Currently, its widespread adoption is still impeded by a number of problems, one of which is difficulty of developing and implementing grid-enabled programs.

The aim of this chapter is two-fold. In the next section we present the state-of-the-art of symbolic and algebraic computations involving parallel and distributed environments as well as the path leading toward grid-enabled CASs. Secondly, we follow up the discussion with two case studies of porting CAS systems to the grid: Maple2g—-grid enabled Maple and MathGridLink-—grid enabled Mathematica.

2. PARALLEL AND DISTRIBUTED SYMBOLIC COMPUTATIONS

Many users utilize CASs as tools performing "small scale" mathematical calculations that would be tedious and error-prone when performed by hand. For them, those systems running on a single processor computer are quite satisfactory. However, there exists a class of users who employ CASs to solve large and very large problems. Here, the solution to the problem may involve, for instance, large scale symbolic computations requiring a significant amount of computational resources, or a combination of symbolic and numerical computations used in the context of multivariable simulation (the CAS environment becomes an "interface" to a set of computational kernels). These users often encounter the limitations of single-processor systems: processor speed and available memory. It is this class of users that could truly benefit from availability of parallel and distributed versions of CASs.

It is well known, that the two main reasons driving the development of parallel computers are: (a) ability to reduce the wall-clock time i.e., the user's waiting time for the solution (problems that are processor bound), (b) ability to solve problems that cannot fit into memory of a "workstation" (problems that are memory bound). An argument has been formulated, that for the CASs it is the latter restriction that is the driving force for parallelization [8]. The argument relies on the following observations. Computation with algebraic terms involves interaction between the amount of available memory and the amount of memory required by the algorithm at any stage of the computation; the input size of a problem may be small, but its memory use in intermediate stages of the computation may grow considerably.

In this context, let us briefly look at algorithms that play a significant role in large-scale symbolic and algebraic computations.

2.1. Parallel Algorithms for Symbolic Computations

Multiprecision integer arithmetic is one of most important fields in symbolic and algebraic computations. It appears, among others, in factorizations [9] or Gröbner base

computations [10]. For parallel arithmetic in finite fields there exists an implementation on a massive parallel processor [11]. Modular integer multiplication [12] and exponentiation [13] have been also implemented on parallel architectures. Systolic algorithms for integer arithmetic were developed and presented in Ref. [10]. Furthermore, systems for multivariate integer arithmetic on distributed memory machines [14] and on the Internet [9] have been developed. Parallel implementation of the Karatsuba algorithm for multiprecision integer multiplication is reported in Ref. [15].

The second class of algorithms that utilize significant amount of computational resources are implementations of polynomial arithmetic. Two categories of algorithms can clearly benefit from multiple resources usage are (a) algorithms that depend on identification of similar terms such as the polynomial addition, and (b) knowledge based algorithms such as the symbolic differentiation or Gröbner base computations [8]. The greatest common divisor is an example of a frequent operation on both integers and polynomials. Parallel GCD algorithms were reported in Refs. [16] and [17]. In Ref. [18] two aspects of parallelism in symbolic computing are discussed: implementation of parallel programs used in factorization of polynomials and automatic generation of parallel codes for finite element analysis. The first aspect illustrates the use of parallel programming to speed up symbolic manipulation, while the second one shows how symbolic systems can help create parallel software for scientific computation.

The third class of algorithms that can benefit from extra memory and processing power availability in parallel systems are the Cramer's rule and the Gaussian elimination with back substitution used to solve sparse systems of linear equations [8].

Finally, the Gröbner base algorithm has application in several mathematical fields. Here, the size of the computation and the irregular data structures required make parallelization an attractive option for improving the algorithm performance. Several parallel implementations of the algorithm have been developed. The one proposed in Ref. [19] consists of parameterized work distribution on shared memory architecture. In Ref. [20] application level threads are used on a distributed memory system. Algorithm implementations on distributed memory systems are reported also in Ref. [8, 21, 22].

While memory-bound algorithms are clearly the most important driving force for the development of parallel CASs, there is one more reason that becomes more important with every new release of each of the major CAS. As indicated above, CASs increase their utility not only through adding new symbolic capabilities, but also by expanding their reach through adding new functionalities, such as visualization or numerical modules. In this way, modern CASs become less of a computational engine and more of a problem solving environment: the CAS is seen as an interface to a number of computational kernels. This change poses a need for addressing the parallelization of processor bound tasks. As examples of such tasks we can list: rendering of images for an animation illustrating the evolution of a system or a multivariable simulation illustrating the solution space of an optimization problem. In these cases, all of the typical problems involved in parallelization of numerical computations appear also in the context of parallel or distributed CAS.

The design and implementation of a robust and scalable CAS relies on the same principles as those applied in other large systems (e.g., modularity, abstraction), but there are also some specific development problems (identified, for instance, in Ref. [8]) that play an important role, when development of a parallel or distributed CAS is concerned like the algorithmic dependence on irregular data which are difficult to be dynamically partitioned, or the complexity of some of ensuing algebraic computations limiting ability to estimate resource requirements. Keeping this in mind, let us now survey the state-of-the-art in parallel and distributed CASs.

2.2. Parallel and Distributed CASs—State-of-the-Art

It is a well known fact that developing completely new parallel or distributed systems, although efficient, in most cases is rather difficult. Moreover, usually only a few parallel algorithms within such a system are fully implemented and tested, making the resulting artifact too limited for practical uses. An alternative approach is to add parallelism to an existing system (consisting of a large number of implemented and tested sequential algorithms). In this case parallelism becomes an added value to the existing environment. Several parallel

and distributed CASs have been developed this way, but based on different requirements and targeted for competing parallel architectures.

Overall, the following developmental strategies were identified in Ref. [8]:

- develop CASs for shared memory architecture;
- develop computer algebra hardware;
- add parallel primitives for communication and cooperation to existing CASs;
- build distributed memory systems based on standard communication middleware;
- build distributed systems for loosely coupled machines or across the Internet.

Let us now proceed with a summary of existing CASs and other symbolic computation systems or tools that have been extended towards parallel and distributed computations. It should be stressed, that even though the authors of this chapter have spent considerable time researching the subject, the list presented in the next sections is far from complete. Our presentation is structured in the following way. We start with three most popular CASs: Maple [23], Matlab [24], and Mathematica [25] and subsequently present a combined list of smaller-scale or niche projects.

Maple. There exist a large number of efforts to extend Maple to parallel and distributed environments and a comprehensive review can be found in Ref. [26]. Here, we present a few selected examples to illustrate the most important approaches taken by various research groups. A message passing interface to port Maple to the Intel Paragon was presented in Ref. [27]: a master-slave approach to distributed scheduling was used to maintain a single node access in interactive use of Maple. A parallel version of Maple running on a network of workstations was reported in [28]: it is a message-passing system with primitives spawn and kill for creating and terminating processes, and procedures send, receive and reply for communication. ‖Maple‖ [29] is a portable system for parallel symbolic computations built as an interface between the parallel programming language Strand and the sequential CAS Maple. Distributed Maple [26] allows the creation of concurrent tasks and have them executed by multiple Maple kernels running on separate networked computers: a Java configuration program starts and connects external computational kernels and schedules concurrent tasks for execution, while a Maple package implements an interface to the scheduler and provides a parallel programming model. The design principles behind PVMaple [30] are very similar to those of the Distributed Maple: 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, while a Maple library implements a set of parallel programming commands available within Maple.

Matlab. More than 20 different versions of parallel Matlab have been developed by different groups of researchers and an overview of them was presented in Ref. [31]. They can be compared according to their process-communication and user interfaces. A significant number of parallel versions of Matlab make use of message-passing for interprocessor communication and provide message-passing interface to the user. Commands like send and receive are based on standard MPI/PVM libraries and utilized in DP-ToolBox [32], MPITB/PVMTB [33], and MultiMatlab [34]. Simple communication functions have been used in Matlab Parallelization Toolkit [35], ParMatlab [36], and PMI [37], while file I/O synchronization functions via a shared file system has been implemented in MatlabMPI [38]. A few parallel Matlabs are designed for shared-memory systems and provide shared-memory programming interfaces—in MATmarks [39], for example, commands are provided for shared variable declaration and process synchronization. Other versions of Matlab are designed to release the user from parallel details by overloading several existing Matlab functions with their parallel versions—for example, Matlab*p [40]. Another approach is to use Matlab compilers, like Conlab [41] or Otter [42]. These compilers can automatically translate a Matlab program into a parallel program written in C. It is worth noting that the Symbolic Math Toolbox provided within Matlab uses the Maple kernel for symbolic computations.

Mathematica. Parallel Computing Toolkit (PCT) [43] is an extension of Mathematica which allows communication between servers using rsh: a typical installation involves

a master kernel to handle inputs, outputs, and scheduling, a license manager to manage licenses and passwords, and one Mathematica kernel per each available node. Based on PCT, gridMathematica [44] was constructed as a parallel computing solution for dedicated clusters facilitating parallel computing within Mathematica, requiring only TCP/IP connectivity. Since it is focused on management of a clusters of heterogeneous machines, a better name for this environment would be clusterMathematica. Distributed Mathematica [45], similar to Distributed Maple, is a system for writing parallel programs in Mathematica allowing to create concurrent tasks and have them executed by Mathematica kernels running on different machines of a network.

Special Libraries and Systems. Threads for parallel symbolic computations on a shared-memory computers were used in parallelizing the SAC-2 CAS, resulting in the ParSac-2 parallel system [46], written in C. It has been used to substantially improve performance of applications such as Gröbner base computations [19] or Karatsuba multiplication. Paclib [47] is a parallel extension of Saclib, a library of C programs for computer algebra derived from the SAC-2 CAS: communication between processes is achieved through access to shared data, and since shared memory limits scalability, very large problems could not have been solved. Givaro [48] is a more recent C++ library that supports parallel programming for arithmetic and algebraic computations with basic algebraic objects, such as vectors, matrices and univariate polynomials.

A special language designed for L-machines was provided by Ref. [49] for parallel programming of computer algebra algorithms: the L-machine consists of a reconfigurable assembly of processors, memory, a bus switch, and a sensor bit used for access rights to the shared memory and for synchronization. Another dedicated machine called FLATS [50] was constructed for large scale CAS applications: it was equipped with special hardware for arbitrary precision arithmetic and parallel hashing in addition to the instruction set for executing Lisp primitives directly by the hardware. In Ref. [51] Lisp was extended with a tool for automatic identification of concurrency: the system accepts a Lisp program, analyses it for available concurrency and generates a program for parallel execution on a multiprocessor comprising a network of workstations. This data flow analyzer can be utilized to analyze a complete Lisp-based CAS such as the Reduce [52] and identify areas that can be parallelized. A different Lisp extension for a distributed network of workstation was given in Ref. [53] where explicit concurrency primitives were provided. Another system called Star/MPI [54] is available for Gnu Common Lisp and GAP [55].

The first version of Cabal [56] written for polynomial algebra, uses the PVM library for parallelization. A later development switched to MPI and several packages for multiprecision integers, matrix algebra and Gröbner base computations were added [57].

MuPad [58], the multi-processing algebra data tool, developed for shared memory parallel machines allows fast access to large databases and functional programming.

Form [59], used in quantum field theory computations, is a program for symbolic manipulation of algebraic expressions that was tuned to handle very large expressions, involving millions of terms. A parallel prototype ParForm, based on MPI, was presented in Ref. [60]: the Form user does not need to modify the sequential versions of the programs, as parallelization being facilitated internally within the environment.

The FoxBox system [61] provides client-server interfaces to several CASs running in distributed computing environments; distribute, wait, kill functions are compliant with the MPI. The DSC system [62] is designed for symbolic computation on network of workstations and across the Internet: a master scheduler distributes tasks based on availability of resources determined through some threshold conditions. It has been used in algebraic computing with large integers and sparse linear systems [63].

2.3. Web-Enabled Systems

In number theory there exist a number of successful Internet projects [64] aiming, among others, at finding large prime numbers, factoring large numbers, computing digits of π, finding collisions on known encryption algorithms etc. A CAS web-wrapper component that can be used by multiple systems was reported in Ref. [65]. Here, shared-memory parallelism

was used to speed up Gröbner basis computations. Another online system for Gröbner basis computations, OGB [66], has been recently deployed.

MapleNet [67] is a software platform to enhance mathematics and related courses over the web. The client uses only the Java virtual machine. To access web pages and the applets associated with them users will connect to the server. It is also the server that manages concurrent Maple instances launched to serve client requests for mathematical computations and display services, and facilitates additional services such as user authentication, logging information, and database access.

WebMathematica [68] offers access to Mathematica applications through a web browser, using standard Java technologies It allows a site to deliver HTML pages that incorporate Mathematica commands. Input can come from HTML forms, applets, JavaScript, and web-enabled applications. It is also possible to send data files to a server for processing. Output can use different formats such as HTML, images, Mathematica notebooks, MathML, XML, PostScript, PDF.

A framework for description and provision of web-based mathematical services was recently designed within the Monet project [69]. Its aim was to demonstrate the applicability of the semantic web to the world of mathematical software. The key to such a framework is the ability to discover services dynamically based on published descriptions which express both their mathematical and non-mathematical attributes. The discovery service and subsequent interactions are mediated by software agents capable of recognizing the criteria which should determine how particular problems are to be solved, and extracting them from the user's problem description. A symbolic solver wrapper was also designed to provide an environment that encapsulates CASs and expose their functionalities through symbolic services. Maple was chosen for the computing engine in the initial implementation, and Axiom was used to demonstrate the ability to incorporate different computational engines without major changes.

3. GRID-ENABLED SYSTEMS

There exist a number of grid-oriented projects that involve CASs. Even though some of these projects have just been initiated, we report their existence and goals for completeness of the overview of the field.

Open source package NetSolve [70] was one of the earliest grid systems developed. Its initial motivation was focused on the usability, portability and availability of existing optimized software libraries for high-performance computing, particularly those for numerical linear algebra. NetSolve is a middleware between desktop systems equipped with simple APIs and the existing services supported by the grid architecture. One of the goals of NetSolve project is to create a system capable of integrating arbitrary computational resources. NetSolve APIs are available for Matlab, Mathematica, and Octave [71]. Version 2.0, released in 2003, introduces GridSolve for interoperability with the grid. GridSolve is a GridRPC [72] based client-server-agent system that enables users to solve complex scientific problems remotely using distributed resources on the grid. When a user submits a problem to the NetSolve agent, the agent searches the grid, chooses a set of suitable services and requests that the problem be solved. After the task is completed the NetSolve agent returns the solution to the user. Load balancing and retry for fault-tolerance are handled automatically by the system. Access to different grids is made possible through proxies. At the present time, proxies for Globus and Condor-G are available. Ninf-G [73] is another GridRPC system implementing the Ninf system on top of the Globus. In version 3, released in 2004, different Ninf client API were build, including one for Mathematica.

The Grid Enabled Numerical and Symbolic Services [74] project, Genss, was initiated in 2004 and follows the ideas formulated in the Monet [69] project. It intends to combine grid computing and mathematical web services using a common open agent-based framework. Thus far research was focused in two areas: (1) matchmaking techniques for advertisement and discovery of mathematical services, and (2) design and implementation of an ontology for symbolic problems.

The Geodise [75] system, implemented within the Matlab environment, is an engineering portal providing grid access to computational fluid dynamics and design optimization tools.

Two different mechanisms are used to submit jobs to computing resources. The first one uses a web service interface to Condor [76]. The second one is implemented as a collection of Matlab functions allowing to run and control jobs on the grid, to archive, query, and retrieve data, or to notify the (mobile) user about the status of the job; submission of jobs to Globus-enabled resources is achieved via Java CoG.

Another on-going project [77] based on Java CoG builds two sets of software tools to enable access to Globus grid resources from Matlab. The first one is a set of wrappers necessary to invoke the CoG batch files directly from the Matlab command line. The second one is a set of Java CoG libraries providing the integration of user codes. Another project, Matlab*g [31] builds a parallel Matlab on a platform-independent grid. It exploits a client-server architecture based on distributed shared memory model. Each server receives a work package, performs computations, and sends results back to the client. First implementation of Matlab*g was based on Alice [78], while a more recent implementation is based on Globus. Another grid extension of Matlab reported in Ref. [79] is based on Matlab*p, mpich-G2, and Globus: it has three components, a server connection manager handling communications, a matrix manager handling the task distribution and a package manager registering the available services.

Grid-Elimino [80] is a recent Java-based computation system for grids based on Globus and Iamc [81]. A master program controls the slave servers, i.e., Elimino instances running as grid services (Elimino is a stand-alone symbolic computation system). The user Java client describes the tasks for each Elimino instance.

The Grid-TLSE [82] project, commenced in 2003, aims to design an expert site for users who are searching for sparse matrix solvers. The administrator interface, called Weaver is used to define, to deploy, and to exploit services over the grid. The web interface, called Websolve, allows a web browser to submit computational requests to a grid by using third-party software such as Matlab, Octave, Scilab, NetSolve, Diet.

Finally, Gemlca [83] is a recent solution to deploy a legacy code application as a grid service without modifying the code. The Gemlca front-end, described in WSDL, offers grid services to deploy, query, submit, check the status of, and get the results back from computational jobs. In order to access a legacy code program, the user executes the grid service client that creates a code instance with the help of a code factory, and the system submits the job to the compute server through Globus.

As can be seen from the above presented summary of the state-of-the-art in parallel and distributed CASs, this area is brimming with activities. In particular, very large body of research is devoted to facilitating symbolic computations on the grid. In order to illustrate in more detail issues involved in porting CASs to the grid we present two case studies based on ongoing research projects involving Maple and Mathematica.

4. CASE STUDY: MAPLE2G

Our analysis of the grid aware CASs presented in the previous section indicates, that any such a system must have at least one of the following facilities (Fig. 1):

— *Ability to accept services from the grid*: the CAS must be capable to augment its facilities with external modules, in particular it should be able to explore computational grid facilities, to connect to a specific grid service, to use it, and to translate its results for the CAS interface. This approach is taken into consideration by the Geodise, Matlab*g, Grid-TLSE, NetSolve/GridSolve.

— *Being a source of grid or web services*: the CAS or some of its facilities must be reachable as grid or web services and allowed to be activated by remote users under appropriate security and licensing conditions; furthermore, deployment of the services must be done in an easy way from the inside of the CAS. This approach is taken into account by the Genss project.

— *Ability to communicate and cooperate over the grid*: several kernels of CASs must be able cooperate within a grid in solving problems; in order to have the same CAS on different computational nodes a "grid-version" must be available; in the case of different CASs, appropriate interfaces between them must be developed and implemented

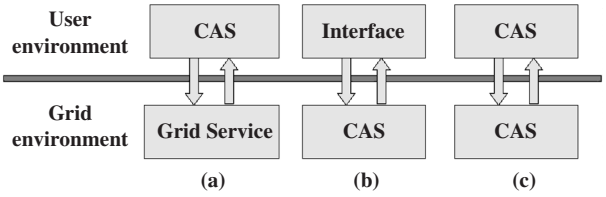

Figure 1. Operating modes between a CAS and a computational grid: (a) CAS as an interface for the grid services; (b) CAS as grid service; (c) multiple CAS kernels on user and grid sides.

or a common language for inter-communication must be adopted. This approach is considered by the Grid-Elimino.

Here we describe a tool supporting the first and the third aspect of grid-enabling of the CAS. More specifically, the Maple2g package allows the connection between Maple and computational grids based on the Globus Toolkit.

The prototype of a grid-enabling wrapper for Maple, consists of two parts (a) a CAS-dependent and (b) a grid-dependent one:

(a) m2g, the library of functions allowing the Maple user to interact with the grid/cluster middleware;
(b) MGProxy, the middleware, a package of Java classes, acting as an interface between the m2g and the grid environment.

The m2g functions are implemented in the Maple language, and they call the MGProxy which accesses the Java CoG API. In this way, any change in the CAS or the grid will be reflected only in *one* part of the proposed system. The CAS-dependent part is relatively simple and can be easily modified to support another CAS or a legacy code.

For the first approach mentioned above we have implemented a minimal set of functions (described in Table 1) allowing access to the grid services.

The m2g package translates functions from the syntax familiar to the Maple user into commands, allowing the initiation and further communication with the MGProxy middleware. The MGProxy is activated from inside the Maple environment by the m2g command m2g_MGProxy_start. The user command(s) originating from the user's Maple interface are send to the MGProxy via a socket interface, when the m2g_getservice and the m2g_jobsubmit functions are invoked. The MGProxy contacts grid services, queries the contacted services, and sends to the Maple interface the results of performed queries. User receives the results through the m2g_receive. Maple commands are passed in the system as strings and the results are presented in the MathML format. m2g_getservice is based on a combination of LDAP and Globus commands.

Several examples of using Maple2g have been presented in Refs. [84] and [85]. Here we present an example of multiplication of two large integers consisting of 10 million, 20 million, and 40 million decimal digits. We have attempted running the Maple code presented in Figure 2 on a PC with a Pentium IV processor running at 1.5 GHz and with 512Mb RAM and encountered the following situations. When Maple 7 was used, the multiplication used the Karatsuba algorithm and the computation time was around 770 seconds for the first pair and would have taken several hours for the subsequent pairs. When Maple 9 was used, the multiplication used the Schönhage-Strassen algorithm and the multiplication time was

Table 1. M2g functions enabling Maple to use grid services.

Function	Description
m2g_connect()	Connects to the grid
m2g_getservice(c, l)	Searches for a service c and give a link to it, retrieve its location l
m2g_jobsubmit(t, c)	Performs a job submission, labeled 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)	Retrieves the results of the submitted job labeled t

```
>#Maple code to measure the time to multiply two big integers
>L:=[2^25,2^26,2^27]: for p in L do a:=2^p-1: s:=time(): a*(a+8): time()-s od;
```

```
// C++ code multbi measuring the time to multiply pairs of large integers
#include <cln/cln.h>
using namespace cln;
int main() { int i; cl_I p=((cl_I)1<<25;
for(i=0;i<3;i++){ a=((cl_I)1<<p)-1; {CL_TIMING; a*(a+8);} p=p<<1;} return(0);}
```

```
> #Maple2g code & results when the grid multiply service is used
> with(m2g);
  [m2g_connect,m2g_exit,m2g_getservice,m2g_jobstop,m2g_jobsubmit,m2g_maple,
   m2g_rank, m2g_recv,m2g_results,m2g_send,m2g_size,m2g_status,
   m2g_MGProxy_end,m2g_MGProxy_start]
> m2g_MGProxy_start(); m2g_connect();
   Grid authorization checked
   Grid connection established
> m2g_getservice("multbi",‘service_location‘);
   [‘&(resourceManagerContact="myri1.info.uvt.ro") (count=1) (label
   "subjob 0") (directory=/home/dana) (executable=/home/dana/multbi)‘]
> m2g_jobsubmit(10,service_location[1]); m2g_results(10);
   Job 10 launched
   real time: 5.012 s, run time: 4.486 s
   real time: 25.565 s, run time: 20.470 s
   real time: 23.541 s, run time: 19.910 s
> m2g_MGProxy_end();
   Grid connection closed
```

Figure 2. Example: Multiplying three pairs of big integers of the order of tens millions decimal digits—Maple code crashes due to memory and time limitations; C++ code based on CLN library and used as grid service to multiply the same big integers; Maple2g code using the grid service to check the multiplication time.

around 9 seconds for the first pair, 21 seconds for the second pair, but the third multiplication could not have been performed (an error “Stack limit reached” occurred, caused by the recursive nature of the algorithm). In the second case, four times more memory was needed to complete the multiplication. A solution to overcome this limit of Maple is that of using an appropriate grid service which can complete the multiplications. We have selected, for testing purpose, the CLN package [86], a C++ library, which allows computations with integers with unlimited precision and which uses the Schönhage-Strassen multiplication for integers larger than 12 thousands decimal digits. The C++ code for the multiplication of the three pairs of big integers is also included in Figure 2. The time of multiplications is less than but comparable to the time obtained with Maple 9, with the difference, that this time we were able to complete the task for the largest integer pair. Time tests have shown that if the CLN package is registered as a grid service, the Maple2g code presented in the same figure can activate it, and the overhead is approximately 16 seconds if the service is local. Additional 12 seconds are required if the service is located remotely within the same fast network (a cluster environment with 2 Gbs connections), and another 18 seconds are required if the service is located remotely in another country.

Table 2. M2g functions for remote process launch/communications.

Function/const.	Description
m2g_maple(p)	Starts p processes MGProxy and p Maple processes
m2g_send(d, t, c)	Sends at the destination kernel labeled 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, c is send to all Maple kernels
m2g_recv(s, t)	Receives from the source kernel labeled s a message containing the results from the a previous Maple command which was labeled with t; when ‘all’ is used in source field, a list is returned with the results from all Maple kernels which have executed the command labeled t
m2g_exit()	Kills all MGProxy processes, shutdowns the twin Maple kernels
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

```
> #Maple code to measure the time to multiply two big integers
> p:=2^25: a:=2^p-1: s:=time(): a*(a+8): time()-s:
```

```
> #Maple2g code
> Karatsuba:=proc(A,B) local N,k,A0,A1,B0,B1,T0,T1,T2,E1,E2:
> N:=max(length(A),length(B)): k:=floor(N/2):
> if(N<10^6) then RETURN (A*B) end:
> with(m2g): m2g_MGProxy_start(): m2g_connect(): m2g_maple(2):
> A0:=irem(A,10^k): A1:=iquo(A,10^k):
> B0:=irem(B,10^k): B1:=iquo(B,10^k):
> m2g_send(1,100,cat("A0:=",A0,":B0:=",B0,": A0*B0;"):
> m2g_send(2,200,cat("A1:=",A1,":B1:=",B1,": A1*B1;"):
> T1:=(A0+A1)*(B0+B1):
> T0:=m2g_recv(1,100): T2:=m2g_recv(2,200): E1:=parse(cat(T2,T0)):
> E2:=parse(cat(T1-T0-T2,substring(convert(10^k,string),2..k+1))):
> m2g_MGProxy_end();
> RETURN(E1+E2) end;
> p:=2^25: a:=2^p-1: s:=time(): Karatsuba(a,a+8): time()-s;
```

Figure 3. Example: Multiplying two integers of 10 millions decimal digits—Maple code used for efficiency comparisons; Maple2g code using Karatsuba algorithm and three Maple kernels to speedup the computation.

Let us now consider grid-based parallel computing involving Maple. Parallel codes using MPICH as the message-passing interface can be easily ported to grid environments due to the existence of a MPICH-G version which runs on top of the Globus Toolkit. On other hand, the latest Globus Toolkit is built in Java, and Java clients are easier to write. This being the case, we selected mpiJava [87] as the message-passing interface between Maple kernels.

In order to follow the third aspect of porting CASs to the grid, Maple2g provides a number of commands for communicating with other Maple kernels. These commands have been summarized in Table 2. Maple2gs parallel computing facilities are similar to those introduced in PVMaple [30]. The user's Maple interface is seen as the master process, while the other Maple kernels are working in a slave mode. Sending commands is possible not only from the Maple interface, but also from one kernel to another, i.e., a user command can contain inside a send/receive command involving slaves.

While additional details and examples of parallel usage of Maple2g can be found in Ref. [88], here, we continue with the integer multiplication example. We consider the case of using three Maple 7 kernels to speedup the multiplication of two integers with 10 million decimal digits. As stated above, the Maple 7 code from Figure 3 running on the PIV 1.5 MHz based PC with 512 Mb RAM uses approximately 770 seconds to finish the multiplication. Using three Maple kernels running the Maple2g code from Figure 3 on a homogeneous cluster consisting of similar PCs connected at 2 Gbs, the running time was reduced to approximately 320 seconds (efficiency of 80%). A 7% loss in efficiency was due to the use of grid environment instead of direct application of MPI). Another loss of 6% of efficiency was registered when one kernel has run remotely (Internet connection between Romania an Austria was involved).

At this stage, Maple2g exists as a demonstrator system. It preserves the regular Maple instruction set while adding several new instructions. Further work is necessary to make it more comprehensive and robust. In the near future we plan intensive tests on grids on a large domain of problems to help guide further development of this 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 [89, 90].

5. CASE STUDY: MATHGRIDLINK

Despite the fact that Mathematica is a widely used programming environment for symbolic computation, grid-aware components for Mathematica have not been developed yet. Only functions for accessing the standard web services are available in Mathematica 5.0, provided as a free add-on. Note that gridMathematica is *not* a grid-aware Mathematica since it is not intended to be used within a computing grid (see subsection 2.2). Mathematica lacks support

for integration of grid services within its standard framework. Neither accessing grid services nor development/deployment of grid services is supported by default. There are some projects aiming to integrate Mathematica into the grid infrastructure, but none of them provide bidirectional access to the grid. Usually, grid-based systems providing Mathematica extensions (e.g., NetSolve/GridSolve, Ninf-G2, etc.) focus only on using Mathematica as a client, using a dedicated service sever specially designed for the particular project. Moreover, the goal of most such systems is to design a proprietary system on top of the grid infrastructure, thus departing from the main idea of grid computing: *creating loosely coupled systems, able to seamlessly interact with each other*.

In this section we present an overview of MathGridLink, a software component designed to act as a middleware between Mathematica and the grid and thus, to the best of our knowledge, the only system involving Mathematica to overcome limitations of, described above, "one-sided" approaches (for a complete description, see Ref. [91]). MathGridLink allows both the development/deployment of Mathematica computational services on the grid and the usage of any existing grid service from within Mathematica.

MathGridLink is built on top of the Globus Toolkit 3 (GT3) and its skeleton is depicted in Figure 4. The system is composed of three main components:

1. MathGrid Service Client (MGSC)—a general purpose grid service client;
2. MathGrid Service Generator (MGSG)—-a component used for grid service generation from Mathematica;
3. MathGrid Service Manager (MGSM)—a specialized Mathematica kernel manager providing remote access to grid services implemented in Mathematica.

Let us now look in some detail at each of these components.

MathGrid Service Client. The first component, MathGrid Service Client (MGSC), is responsible for the invocation of grid services from Mathematica. There are two known main methods for accessing a remote service: (i) by using specific client stubs (generated from the WSDL description of a given service); and (ii) by using a general purpose client. The former involves generation of a client stub and requires writing specialized code for each service we intend to use. The latter method is more flexible, allowing access to any service without the need of writing any additional code. MGSC implements the second approach.

MGSC is implemented in Java and the interface of the invoked grid service is adapted so that the user has full access to the requested service's methods and their usage description. Furthermore, access to any grid service is transparent, in the sense that the access is made only by invocation of eight simple Mathematica functions, presented in Table 3.

Some computational services perform heavyweight computations requiring a significant time for their completion. When such operations are performed remotely, client application has to wait until the results are ready. In contrast, some "informative" grid services require a very short execution time (e.g., a grid service which informs about the current temperature

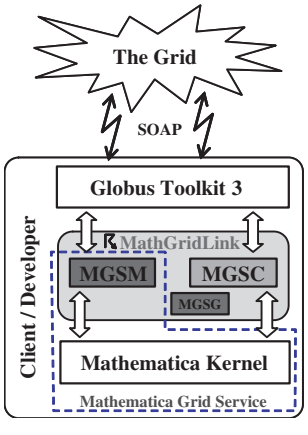

Figure 4. General MathGridLink overview.

Table 3. Mathematica functions provided by MGSC for usage of remote grid services.

Function	Description
MGUseService[*s*]	Connects to the grid service *s*, creates a new service instance and fetches the information about the deployed methods. If successfully completed, it returns the reference to the newly created service instance.
MGCall[*sr*, *m*, *args*]	Performs a blocking call of the method *m* for the service *sr* using the arguments *args*. It returns the result.
MGModelessCall[*sr*, *m*, *args*]	Initiates a non-blocking call of the method *m* for the service *sr* using the arguments *args*. It does not return anything.
MGWait[*sr*]	Blocks the current Mathematica kernel until the service instance referenced by *sr* finishes a previous MGModelessCall call and the result is ready.
MGGetResult[*sr*]	Gets the result of the previous call to this service instance. During a non-blocking execution, it returns Null.
MGDestroy[*sr*]	Destroys the service instance referenced by *sr*.
MGList[*sr*]	Returns a list of available methods for service instance referenced by *sr*.
MGHelp[*sr*, *m*]	Returns the invocation details for method *m* of the service referenced by *sr*.

for a specific city). Based on this observation, a general-purpose client has to be able to deal with these different behaviors. MGSC supports two different kinds of service invocation:

1. *blocking call*: The Mathematica kernel cannot do anything else until the service has sent the result;
2. *non-blocking call*: A parallel implementation of service invocations using a semaphore-based synchronization.

To illustrate the case of a blocking call, assume that a grid service is running at *http://myweather.org/Temperature* implementing a method named `getTemperature` that accepts as parameter a zip-code. In order to find current temperature, a typical blocking call will involve the following skeleton:

```
...
GSH = ''http://myweather.org/Temperature'';
...
s = MGUseService[GSH];
Print[MGCall[s, ''getTemperature'', ''305-8573'']];
...
MGDestroy[s];
...
```

Since the execution time of some computational services may take several minutes, or even hours, the non-blocking call is, obviously, a more sensible one. Let us assume that we have a grid service running at *http://myservice.org/Prime* which implements a method named `findPrimes` accepting a number as the parameter and returning a list of all prime numbers smaller than the given number. If the number is very large, the computation time will be significant. If, while the computations are being performed, we want to use the Mathematica's front end, or to invoke another service, we could use the following non-blocking call of the service:

```
...
GSH = ''http://myservice.org/Prime'';
...
arg = 983457349976239645247 9;
...
s = MGUseService[GSH];
MGModelessCall[s, ''findPrimes'',arg];
(* do some other computations *)
...
MGWait[s];
Print[MGGetResult[s]];
...
MGDestroy[s];
...
```

As a result, the use of the Mathematica Kernel is not blocked, and other operations may be executed while the remote service completes our request.

MathGrid Service Generator. The second component, MathGrid Service Generator (MGSG), is responsible for automatic generation of grid services. The MGSG is written in Java. It is callable from Mathematica, and can be used for generating a grid service, even if the user has no knowledge of Java and/or the grid. A Mathematica grid service could be generated by using the five functions summarized in Table 4.

From the point of view of the Mathematica user, the process of creating a grid service consists of 2 parts:

(a) creation of a Mathematica package containing the Mathematica implementation of the service;
(b) generation of a grid service for linking others to this Mathematica implementation.

Note that MathGridLink does not translate the implementation of the service code into a Java code in order to deploy it on the grid. Instead, it creates a connecting "bridge" between the grid and the Mathematica kernel. In this way, the service will be able to take advantage of Mathematica computational engine while being available to other systems and programming languages as well.

The process of generating a grid service can be seen as a two-phase process consisting of the *configuration phase* and the *service generation phase*. During the configuration phase, minimal information about the desired grid service has to be supplied:

1. the name of the service to generate;
2. the name of the Mathematica package containing the implementation;
3. the methods to deploy: The Mathematica package implementing our service methods may have several exported functions. However, only the explicitly specified functions will be made available to the others through the generated grid service.

Once the information is supplied, the grid service is generated and the process is displayed in the Java console. The MGSG creates the grid web service interface (based on the developer's specifications of exported Mathematica functions), the server stub, the WSDL and the WSDD service description files, and generates ready-to-deploy Grid ARchive (GAR). The resulting service will deploy the specified Mathematica functions as grid service methods, providing a transparent remote access to the Mathematica kernel. The GAR file contains all files and classes required for deployment, except the Mathematica package which has to be provided together with the GAR file. The service may be deployed on any system with GT3 and Mathematica installations.

To illustrate the process let us consider the following example. Let us assume that a Mathematica package named NumericalFunctions, implements the `LargestPrimeGap` function for finding the largest difference between consecutive primes not exceeding a given number N. To generate and deploy a service named MathService and make this `LargestPrimeGap` function available to the grid as *myPrimeGap*, we can use the following Mathematica code:

```
...
InitMGSG[];
MGSGSetServiceName[''MathService''];
```

Table 4. Mathematica functions provided by MathGrid Service Generator (MGSG).

Function	Description
InitMGSG[]	Initializes the MGSG engine.
MGSGSetServiceName[n]	Sets the name of the service to n.
MGSGSetMathematicaPackage[p]	Specifies the Mathematica package containing the implementation of the grid service as Mathematica functions.
MGSGAddMethod[f, d]	Adds Mathematica function f to the list of methods deployed by the service. We have to specify in d the invocation details in a Java-like format.
MSGSGenerate[]	Generates, compiles and packs the service as a standard GAR (Grid ARchive) file.

```
MGSGSetMathematicaPackage[''NumericalFunctions''];
MGSGAddMethod[''LargestPrimeGap'', ''int myPrimeGap(int x)''];
MGSGGenerate[];
...
```

These commands will generate a GAR file containing all that is needed for deploying the service.

MathGrid Service Manager. The third component, MathGrid Service Manager (MGSM), is responsible for the remote access to the Mathematica grid service and it can be configured to use single or multiple Mathematica kernels for a single grid service. MGSM provides concurrent access to the Mathematica kernels. The user persistency problem that occurs in concurrent access to Mathematica kernels is solved by imposing a different context for each user.

Limitations of the Current Version. Two important design and implementation issues are pending: GT3 Security and GT3 Index Service. At present, only unrestricted services can be used and deployed, and the access to the grid service can be performed only by knowing the GSH (Grid Service Handle) of the service.

6. CONCLUSIONS

In this chapter we have tried to achieve two objectives. First, to introduce recent developments in symbolic computations, in particular, computer algebra systems. Here, we were particularly interested in initiatives leading to porting CASs to parallel and distributed computers as well and making them Web and grid enabled. We have presented a rather extensive list of past present and future projects attempting to reach these goals. Second, to illustrate issues involved in porting CASs to the grid, we have presented two case studies involving some of the more popular CASs Maple and Mathematica. The most important findings can be summarized as follows:

- a growing interest in symbolic computations and computer algebra systems can be observed;
- application of CASs to solution of large problems often demands application of parallel and/or distributed computers;
- porting CASs to parallel and distributed computers is not a trivial task;
- number of existing and freshly started projects indicates fast growing interest in porting CASs to the Web and to the grids in particular;
- real world applications of CASs on grids are in very early phases and efficiency of proposed tools will have to be further investigated and improved;
- while most of project involving CASs and grids are in very early stages, existing tools are mature enough to allow experimental work to be initiated and this is this type of work that is required to lay ground for future developments.

REFERENCES

1. J. von zur Gathen and J. Gerhard, "Modern Computer Algebra." Cambridge Press, 2003.
2. Cain, http://research.mupad.de/CAIN/.
3. Sac, http://www.symbolicnet.org/systems/Systems.html.
4. SAL, http://www.sai.msu.su/sal/A/1/index.shtml.
5. L. Bernardin, "A Review of Symbolic Solvers" (1996), http://www.inf.ethz.ch/personal/bernardi/solve/review_A4.ps.
6. P. Stewart, "Symbolic Computation—A Review" (1992), http://www.bham.ac.uk/ctimath/reviews/nov92/symbol.pdf.
7. M. Wester, in "A Critique of the Mathematical Abilities of Computer Algebra Systems" (M. Wester, Ed.), CASs—A Practical Guide. Wiley, 1999, http://math.unm.edu/~wester/cas_review.
8. M. Matooane, Parallel Systems in Symbolic and Algebraic Computation, Ph.D. Thesis, Cambridge, 2001, http://www.cl.cam.ac.uk/TechReports/UCAM-CL-TR-537.pdf.
9. R. P. Brent, "Some Parallel Algorithms for Integer Factorisation" (P. Amestoy, Ed.), In Proceedings of Euro-Par'99, LNCS 1685, Springer, 1999, http://web.comlab.ox.ac.uk/oucl/work/richard.brent/pd/rpb193sp.pdf.

10. T. Jebelean, in "Integer and Rational Arithmetic on MasPar" (J. Calmet, Eds.), pp. 162–173, Design and Implementation of Symbolic Computation Systems, LNCS 1128, Springer, 1996.
11. E. Sibert, H. F. Mattson, and P. Jackson, "Finite Field Arithmetic Using the Connection Machine" (R. E. Zippel, Ed.), pp. 51–61, In Proceedings of the 2nd Workshop CA&Parallelism, LNCS 584, Springer, 1990.
12. M. Diab, "Systolic Architectures for Multiplication over Finite Fields GF2m" (S. Sakata, Ed.), pp. 329–340, In Proceedings of the 8th International Conference on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, LNCS 508, Springer, 1990.
13. M. Morii and Y. Takamatsu, "Exponentiation in Finite Fields Using Dual Basis Multiplier" (S. Sakata, Ed.), pp. 354–366, In Proceedings of the 8th International Conference on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, LNCS 508, Springer, 1990.
14. J. L. Roch, "PAC: Towards a Parallel Computer Algebra Co-processor" (Della Dora and J. Fitch, Eds.), pp. 33–50, Computer algebra and parallelism, Academic Press, 1989.
15. T. Jebelean, M. Drăgan, D. Ţepeneu, and V. Negru, Technical Report 00–42, Parallel algorithms for practical multiprecision arithmetic using the Karatsuba method, RISC, 2000, ftp://ftp.risc.uni-linz.ac.at/pub/techreports/2000/00-42.ps.gz.
16. R. Kannan, G. Miller, and L. Rudolph, "Sublinear Parallel Algorithm for Computing the Greatest Common Divisor of Two Integers," SIAM Journal Comp. 16, 7–16 (1987).
17. P. L. Montgomery, "An FFT Extension of the Elliptic Curve Method of Factorization," Ph.D. Thesis, University of California, 1992, ftp://ftp.cwi.nl/pub/pmontgom/ucladissertation.ps.gz.
18. P. S. Wang, "Symbolic Computation and Parallel Software, 1991, http://icm.mcs.kent.edu/reports/1991/ICM-9109-12-ab.pdf.
19. B. Amrhein, O. Gloor, and W. Küchlin, "A Case Study of Multithreaded Gröbner Basis Completion," In Proceedings of the ISSAC96, ACM Press, 1996, http://www-sr.informatik.uni-tuebingen.de/projects/pareqs/issac96.ps.
20. S. Chakrabarti and K. Yelick, "Implementing an Irregular Application on Distributed Memory Multiprocessor," In Proceedings of the 4th ACM SIGPLAN POPP'93, ACM Press, 1993, p. 169–178, http://www.cs.berkley.edu/~yelick/soumen/grobner-ppopp93.ps.
21. C. G. Ponder, in "Evaluation of Performance Enhancements in Algebraic Manipulation Systems" (J. Della Dora and J. Fitch, Eds.), pp. 51–73, Computer Algebra and Parallelism, Computational Mathematics and Applications, Academic Press, 1989.
22. A. A. Reeves, "A Parallel Implementation of Buchberger'S Algorithm over z_p for $p \geq 31991$," J. Symb. Comp. 26, 229–244 (1998).
23. Maple, http://www.maplesoft.com/.
24. Matlab, http://www.mathworks.com/.
25. Mathematica, http://www.wolfram.com/products/mathematica/.
26. W. Schreiner, C. Mittermaier, and K. Bosa, in "Distributed Maple–Parallel Computer Algebra in Networked Environments," J. Symb. Comp. 35, 305–347 (2003).
27. L. Bernardin, "Maple on a Massively Parallel, Distributed Memory Machine" (M. Hitz and E. Kaltofen, Eds.), pp. 217–222, In Proceedings of PASCO'97, ACM Press, 1997.
28. S. M. Watt, "A System for Parallel Computer Algebra Programs" (B. F. Caviness, Ed.), pp. 537–538, In Proceedings of Eurocal'85, LNCS 204, Springer, 1985.
29. R. Pirastu and K. Siegl, "Parallel Computation and Indefinite Summation: A ‖Maple‖ Application for the Rational Case," J. Symb. Comp. 20, 603–616 (1995).
30. D. Petcu, "PVMaple—A Distributed Approach to Cooperative Work of Maple Processes" (J. Dongarra et al., Ed.), pp. 216–224, In Proceedings of EuroPVM/MPI, LNCS 1908, Springer, 2000.
31. Y. Chen, and S. Fong Tan, "Matlab*g: A Grid-Based Parallel Matlab," SMA, NUS, Singapore, 2002, http://ntu-cg.ntu.edu.sg/Grid_competition/report/grid-9.pdf.
32. DP-Toolbox, http://www-at.e-technik.uni-rostock.de/dp/.
33. MPI/PVM Toolbox for Matlab (MPITB/PVMTB), http://atc.ugr.es/javier-bin/mpitb_eng and http://atc.ugr.es/javierbin/pvmtb_eng.
34. MultiMatlab, http://www.cs.cornell.edu/Info/People/lnt/multimatlab.html.
35. Matlab Parallelization Toolkit, http://www.mathworks.com/matlabcentral/fileexchange/ (utilities, distributed computing).
36. ParMatlab, http://petrydpc.itm.mh.se/tools/.
37. Pmi, ftp://ftp.mathworks.com/pub/contrib/v5/tools/PMI.
38. MatlabMPI, http://arXiv.org/abs/astro-ph/0107406.
39. MATmarks, http://polaris.cs.uiuc.edu/matmarks/.
40. R. Choy, Matlab*p 2.0: Interactive Supercomputing Made Practical, MS Thesis, EECS, MIT, 2002, http://www.cs.ucsb.edu/~ gilbert/cs290iSpr2003/ChoyThesis.ps.
41. Conlab, http://www.cs.umu.se/research/conlab/.
42. Otter, http://www.cs.orst.edu/~ quinn/papers/hpdc7.ps.
43. Parallel Computing Toolkit, http://www.wolfram.com/products/applications/parallel/.
44. gridMathematica, http://www.wolfram.com/products/gridmathematica/.
45. C. Pau and W. Schreiner, Distributed Mathematica, 2000, www.risc.uni-linz.ac.at/software/.
46. W. Küchlin, "Parsac-2: Parallel Computer Algebra" (J. Fleisher, J. Grabmeier, F. Hehl, and W. Küchlin, Eds.), Computer Algebra in Science and Engineering, World Scientific, 1995, http://www-sr.informatik.uni-tuebingen.de/projects/pareqs/zif94.ps.

47. H. Hong, A. Neubacher, and W. Schreiner, "The design of the SacLib/PacLib Kernels" (A. Miola, Ed.), In Proceedings DISCO'93, Gmunden, Austria, LNCS 722, Springer, Berlin, 1993, http://www.risc.uni-linz.ac.at/people/schreine/papers/disco93.ps.gz.
48. Givaro, http://www-lmc.imag.fr/Logiciels/givaro/.
49. B. Buchberger, "The L-Machine: An Attempt at Parallel Hardware for Symbolic Computation," In Proceedings of AAECC-3, LNCS 229, Springer-Verlag, 1986, pp. 333–347.
50. E. Goto, "Design of a Lisp machine FLATS," In Proceedings of ACM Symposium on Lisp and Functional Programming, Pittsburgh, 1982, pp. 208–215.
51. J. Marti and J. Fitch, "The Bath concurrent LISP machine," In Proceedings of European Computer Algebra Conference on Computer Algebra, LNCS 162, 1983, pp. 78–90.
52. Reduce, http://www.uni-koeln.de/REDUCE/.
53. R. H. Jr. Halstead, "Parallel Symbolic Computing: Languages, Systems, and Applications," In Proceedings of US/Japan Workshop, Cambridge, LNCS 748, 1992.
54. G. Cooperman, "Star/MPI: Binding a Parallel Library to Interactive Symbolic Algebra Systems," In Proceedings of ISSAC'95, ACM Press, 1995, pp. 126–132.
55. GAP, http://www-gap.dcs.st-and.ac.uk/~ gap/.
56. A. Norman and J. Fitch, "Cabal: Polynomial and Power Series Algebra on a Parallel Computer" (M. Hitz and E. Kaltofen, Eds.), pp. 196–203, In Proceedings of PASCO'97, ACM press, 1997.
57. M. Matooane and A. Norman, "A Parallel Symbolic Computation Environment: Structures and Mechanics" (P. Amestoy, Eds.), pp. 1492–1495, In Proceedings of Euro-Par, LNCS 1685, Springer, 1999.
58. MuPAD, http://research.mupad.de/.
59. Form, http://www.nikhef.nl/~ form/.
60. D. Fliegner, A. Retey, and J. A. M. Vermaseren, "Parallelizing the Symbolic Manipulation Program FORM," 1999, http://arXiv.org/abs/hep-ph/9906426.
61. A. Diaz and E. Kaltofen, "FoxBox: A System for Manipulating Symbolic Objects in Black Box Representation" (O. Gloor, Ed.), pp. 30–37, In Proceedings of ISSAC'98, ACM Press, 1998.
62. A. Diaz, E. Kaltofen, K. Schmitz, and T. Valente, "DSC: A System for Distributed Symbolic Computation" (S. M. Watt, Ed.), pp. 323–332, In Proceedings of ISSAC'1991, ACM Press, 1991.
63. A. Diaz, M. Hitz, E. Kaltofen, A. Lobo and T. Valente, "Process Scheduling in DSC and the Large Sparse Linear Systems Challenge," J. Symb. Comp. 19, 269–282 (1995), and LNCS 722, p. 66–80, ftp://ftp.cs.rpi.edu/pub/kaltofen/DSC/DISCO93/jsc.ps.
64. Internet-Based Distributed Computing, http://www.aspenleaf.com/distributed/ap-math.html.
65. A. Weber, W. Küchlin, B. Eggers, and V. Simonis, Parallel Computer Algebra Software as a Web Component, http://www.cs.ucsb.edu/conferences/java98/papers/algebra.pdf.
66. M. McGettrick, OGB: Online Gröbner Bases, Technical report NUIG-IT-251103, National University of Ireland, Galway, 2003, http://grobner.it.nuigalway.ie.
67. MapleNet, http://www.maplesoft.com/maplenet/.
68. webMathematica, http://www.wolfram.com/products/webmathematica/.
69. Monet, http://monet.nag.co.uk.
70. S. Agrawal, J. Dongarra, K. Seymour, and S. Vadhiyar, in "NetSolve: Past, Present, and Future—A Look at a Grid Enabled Server" (F. Berman, Ed.), p. 613–622, Making the Global Infrastructure a Reality, Wiley, 2003, http://icl.cs.utk.edu/news_pub/submissions/netsolve-ppf.pdf.
71. Octave, http://www.octave.org/.
72. H. Nakada, S. Matsuoka, K. Seymour, J. Dongarra, C. Lee and H. Casanova, "GridRPC: A Remote Procedure Call API for Grid," www.eece.unm.edu/~ apm/docs/APM_GridRPC_0702.
73. Ninf, http://ninf.apgrid.org/.
74. Genss, http://genss.cs.bath.ac.uk/index.htm.
75. Geodise, http://www.geodise.org/.
76. Condor, http://www.cs.wisc.edu/condor/.
77. CoG Kit Matlab Page, http://www-unix.globus.org/cog/matlab/index.php.
78. ALiCE, http://www.comp.nus.edu.sg/~teoym/alice.htm.
79. I. Patel and W.Mohiuddin, "Matlab*p on the Grid," 2003, pompone.cs.ucsb.edu/~imran/matlabpg/.
80. Y. Wu, W. Liao, P. Wang, D. Lin, G. Yang, "An Internet Accessible Grid Computing System: Grid-Elimino," In Proceedings of IAMC, 2003, www.symbolicnet.org/conferences/iamc03/grid.pdf.
81. P. S. Wang, S. Gray, N. Kajler, D. Lin, and W. Liao, "IAMC Architecture and Prototyping," In Proceedings of ISSAC, London, Ontario, 2001, http://icm.mcs.kent.edu/research/IAMC.icm/issac01.pdf.
82. GridTLSE, http://www.enseeiht.fr/lima/tlse.
83. Gemlca, http://www.cpc.wmin.ac.uk/GEMLCA.
84. D. Petcu, D. Dubu, and M. Paprzycki, "Towards a Grid-aware Computer Algebra System" (M. Bubak et al., Eds.), pp. 490–495, In Proceedings of ICCS'04, LNCS 3036, Springer, 2004.
85. D. Petcu, D. Dubu, and M. Paprzycki, "Extending Maple to the Grid: Design and Implementation" (J. Morrison, Ed.), pp. 209–216, In Proceedings of ISPDC'04, IEEE Computer Press, Los Alomitos, 2004.
86. B. Haible, "CLN, a Class Library for Numbers," 2004, ftp://ftp.ilog.fr/pub/Users/haible/gnu/cln-1.18.tar.bz2.
87. mpiJava, http://www.npac.syr.edu/projects/pcrc/HPJava/mpiJava.html.
88. D. Petcu, D. Dubu, and M. Paprzycki, "Grid-Based Parallel Maple" (D. Kranzmüller et al., Eds.), pp. 215–223, In Proceedings of EuroPVM/MPI'04, LNCS 3241, Springer, 2004.

89. A. Gilbert, A. Abraham, and M. Paprzycki, "A Framework for Ensuring Data Integrity in Grid Environments," In Proceedings of ITCC'04, IEEE Computer Society, 2004, Vol. 1, pp. 435–439.
90. A. Gilbert, J. Thomas, and I. Jonyer, "Modeling Work Flow in Hierarchically Clustered Distributed Systems," In Proceedings of PDPTA04, 2004, http://www.cs.okstate.edu/˜jonyer/pub/pdpta04.pdf.
91. D. Ţepeneu and T. Ida, "MathGridLink—Connecting Mathematica to the Grid," In Proceedings of IMS'04, Banff, Alberta, Canada, 2004.