PVMaple: a Distributed Approach to Cooperative Work of Maple Processes

Dana Petcu

Western University of Timişoara, B-dul V.Pârvan 4, 1900 Timişoara, Romania,
petcu@info.uvt.ro,
WWW home page: http://www.info.uvt.ro/~petcu

Abstract. In this paper we present the structure and functionality of a prototype system, namely PVMaple, for cooperative Maple applications running on networked workstations in a message passing environment supplied by PVM.

1 Introduction

Recent developments in parallel distributed systems have resulted in increased use of suitable environments, like PVM, which make the message-passing programming to solve complex problems easier and faster for users. PVM framework allows the user to write his applications as a collection of cooperative tasks.

Computer algebra systems (CAS) can be successfully used in prototyping sequential algorithms for symbolic or numeric solution of mathematical problems. Maple, widely used environment for scientific computing, is such a CAS. Constructing prototypes for parallel algorithms in Maple is an actual challenging problem.

Several successful attempts have been made to combine Maple with parallel or distributed computation features. ||Maple|| [9], developed at the beginning of the 1990s, is a portable system for parallel symbolic computations built as an interface between the parallel declarative programming language Strand and the sequential CAS Maple. Sugarbush [2] combines the parallelism of C/Linda with the Maple V Release 3 kernel. In [1] porting Maple kernel to the Intel Paragon family of massively parallel distributed memory machines is described. A number of Maple kernels running on different machines of a local network can communicate also by a simple mechanism based on reading and writing on shared files in a global network file system [10]. FoxBox [3] provides an MPI-compliant distribution mechanism that allows for parallel and distributed execution of FoxBox programs; it has a client/server style interface to Maple. The most recent Distributed Maple ([7] and [8]) 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 different machines of a network. The system can be used in any network environment where Maple and Java are available. It provides message passing facilities via a global heap.

We present a prototype system allowing to study the issue of interconnecting PVM and Maple, namely Parallel Virtual Maple (shorter PVMaple). Its aim is to interface the flexible process and virtual machine control from the PVM system with several Maple processes thus allowing Maple applications the ability to inter-operate transparently across multiple heterogeneous hosts. It provides facilities for post-execution analysis of the dynamic behavior of a session. The design principles are very similar to that of Distributed Maple.

2 PVMaple inside

PVMaple system proposes an extension of Maple capabilities to distributed computations for workstations grouped into a Parallel Virtual Machine. The user interacts with the system via the text oriented Maple front-end. PVMaple can be used in any network environment where Maple and PVM are available. The necessary packages are enumerated in Table 1.

Table 1. PVMaple components and dependencies

Component	Format	Machine	Restrictions
Package pvm.m	Maple	local	to be readed before any pvm[ ]
Command messenger	Binary	local	to be started after pvmd
File with binary paths	Text	all	in the same path
Maple V	Binary	all	at least Release 4
Pvmd	Binary	all	at least version 3.4

Figure 1 indicates the active processes on each machine of the PVMaple system. The communication routes are represented by arrows. The command-messenger ensures the inter-communication between the Maple processes via the PVM environment. Initialization of a Parallel Virtual Maple session, creation of Maple processes and inter-processes communications are provided by pvm.m package.

Table 2 enumerates the functions and constants which are currently included in the pvm.m package. Four separate issues are here specially addressed: process start-up facilities, Maple process identifiers, transparent message passing, and post-execution analysis of Maple session behaviour. A Maple process is identified by a tuple pair [machine_name,processes_id]. Another external Maple process can communicate with the first mentioned one by using his tuple via the package's function calls pvm[send] and pvm[receive]. Note that the Maple processes cannot communicate directly. A Maple send command is registered by the associated command messenger which will send a message (via pvmds) to the destination command messenger which at his turn informs the associate Maple process about the incoming message.

Figure 2 shows the time sequence in which the processes are activated when PVMaple is started and also the effects of different commands from the pvm.m package (via pvmd and pvm-functions implemented in the command messenger).

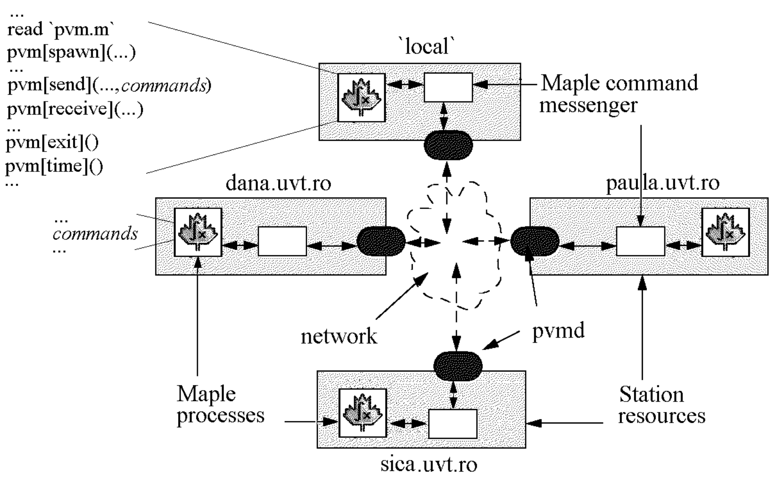

Fig. 1. Processes and inter-communication routes

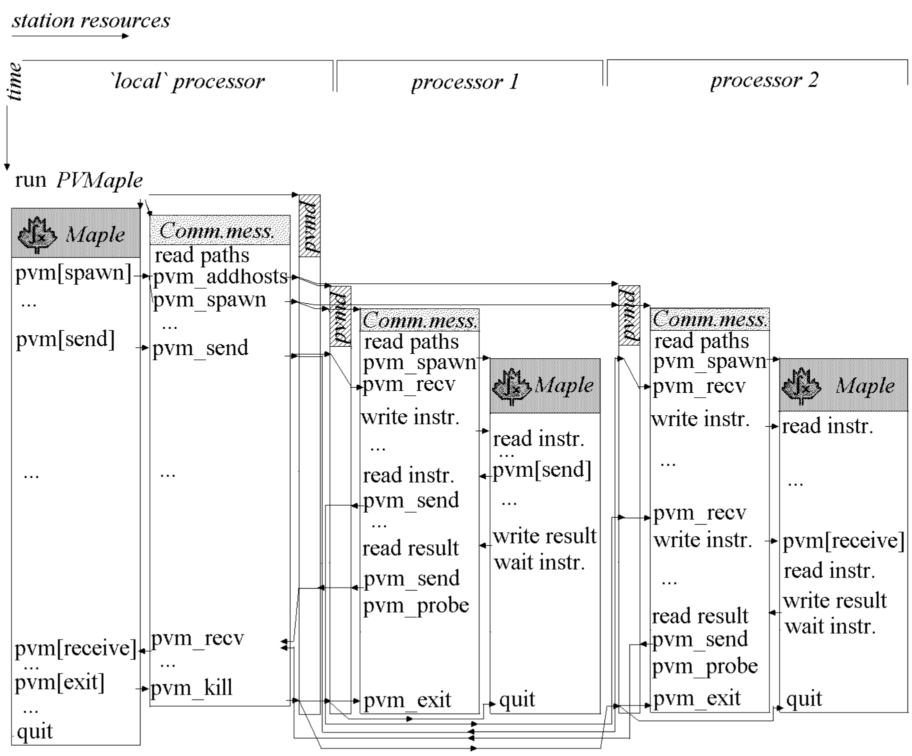

Fig. 2. Communications and components inter-dependencies

Table 2. Functions and constants

Function	Synopsis	Meanings/Parameters
spawn	pvm[spawn](*sequence*)	create Maple processes; *sequence* is a sequence of elements like [*station_name,processes_no*]
send	*int*:=pvm[send](*destin,message*)	send Maple commands; *int* is the message identifier, *destin* can be 'all' for all processes or [*station_name,process_id*] *message*: a string with Maple commands
receive	*list*:=pvm[receive](*mesid,source*)	receive processes results; *list* store results returned by each process, *mesid* is the *int* given by a send or 'all' *source*: 'all' for all processes or [*station_name,process_id*]
exit	pvm[exit]()	return a success or fail message
settime	pvm[setttime]()	start time registration
time	pvm[time]()	show a time graphic
version	pvm[version]()	version message

Constant	Synopsis	Meaning
ProcId	*integer*:=pvm[ProcId]	process identifier on a particular station
MachId	*integer*:=pvm[MachId]	station number into the virtual machine
TaskId	*integer*:=pvm[TaskId]	process identifier into the virtual machine
Tasks	*list*:=pvm[tasks]	list of process identifiers [*station_name,process_id*]

3 Examples

In this section we evaluate the behaviour of our prototype. The tests have been run in a network composed of five identical PC Pentium 250 MHz with 64 MB of RAM running Windows '95. The PVM 3.4 version for Windows '95 [4] was used for message passing.

Table 3 presents three examples of command sequences written in Maple using the pvm.m package. In the first one we can see two kind of a message send command: between the main Maple process (local process) and a remote Maple process (located on machine dana), and between a remote Maple process (located on machine dana) and another remote Maple process (located on machine paula). The Maple command sequences which will be executed on remote processes are those included between pairs of back quotes (Maple commands starting with M and N). The graphical result of the pvm[time] command is presented in Figure 3. The active time periods of Maple processes and auxiliary command messengers are indicated by horizontal lines. The total active time (between a pvm[settime] and a pvm[exit] command) of a Maple process is expressed in percent, so that the user can easily estimates the load balance of the distributed algorithm, the delays and the idleness sources.

Table 3. Examples of distributed commands wrote in Maple

Example 1: communication between processes

> restart; read ‘pvm.m‘;	*load the PVMaple functions*
> pvm[settime]();	*start time chronometer*
> pvm[spawn]([dana,1],[paula,1]);	*start two tasks*
Active tasks: [[‘local‘, 1], [dana, 1], [paula, 1]]	*pvm[tasks] values*
> M:=pvm[send]([dana,1],	*send command to the first task:*
‘pvm[send]([paula,1],“global S;	*send a message to another task,*
S:=2“); S:=5;‘):	*and finish with an assignation*
pvm[receive](M,[dana,1]);	*receive last command's result*
[5]	*1st task last command's result*
> N:=pvm[send]([paula,1],‘S:=3;	*send command to the second task:*
pvm[receive]([dana,1],‘all‘); S+10;‘):	*receive message from task 1.*
pvm[receive](N,[paula,1]);	*receive last command's result*
[12]	*2nd task last command's result*
> pvm[exit]();	*kill the tasks and stop the time*
PVMaple quitted	*PVMaple stopped*
> pvm[time]();	*show the time graph*

Example 2: matrix-vector multiplication

```
> restart: with(linalg): read ‘pvm.m‘: pvm[settime] ():
> pvm[spawn] ([dana,1],[sica,1],[bubu,1],[paula,1]):
Active tasks: [[‘local‘, 1], [dana, 1], [sica, 1], [bubu, 1], [paula,1]]
> #variant: pvm[spawn] ([dana,4]]): 4 tasks on the same processor
> n:=500: p:=5: r:=n/p: x:=randmatrix(n,1): X:=mat2str(x):
> todo:=‘with(linalg): r:=‘.r.‘: n:=‘.n.‘: x:=‘.X.‘:‘:
> todo:=“.todo.‘ A:=randmatrix(r,n): multiply(A,x);‘: s:=pvm[send] (‘all‘,todo):
> z:=array(1..n): A:=randmatrix(r,n): u:=multiply(A,x):
> v:=pvm[receive] (s,‘all‘):
> for i from 1 to n-r do z[i]:=v[iquo(i-1,r)+1][irem(i-1,r) +1,1] od:
> for i from n-r+1 to n do z[i]:=u[i-n+r,1] od:
> pvm[exit](): pvm[time] ();
```

Example 3: parallel ODE method implementation (one iterative step)

```
> Hammer_method_onestep_pvm:=proc(y,h,t)
> local h0,h1,h2,h3,K12,i, C,E,c, R, S,j,K1,K2,K3,K4; global k1,k2,k3,k4;
> vect2str(y,h1); num2str([h,t,n],[h2,h3,h0]);
> S:=pvm[send] ([dana,1],‘global n,y,h,t,k1,k2: n:=‘.h0.‘:y:=vector(n,‘.h1.‘):
h:=‘.h2.‘: t:=‘.h3.‘: fsolve( vect2set( k1-f( t+h*c[1],y+h* (a[1,1]*k1+a[1,2]*
k2)))) union vect2set( k2-f( t+h*c[2], y+h*(a[2,1]*k1+a[2,2]*k2)))), vect2set(
k1) union vect2set(k2)): assign(''): k1,k2;‘):
> k3:=‘k3‘: k4:=‘k4‘: k3:=vector(n): k4:=vector(n): R:=fsolve( vect2set(k3-f(
t+h*c[3], y +h*(a[3,3]*k3+a[3,4]*k4)))) union vect2set( k4-f( t+h*c[4], y+
h*(a[4,3]*k3+a[4,4]*k4)))),vect2set(k3) union vect2set(k4));
> K12:=pvm[receive] (S,[dana,1]); k1:=K12[1]: k2:=K12[2];
> assign(R): list2vect([k1,k2,k3,k4],[K1,K2,K3,K4]):
> evalm(y+h*(b[1]*K1+b[2]*K2+b[3]*K3+b[4]*K4)); end:
```

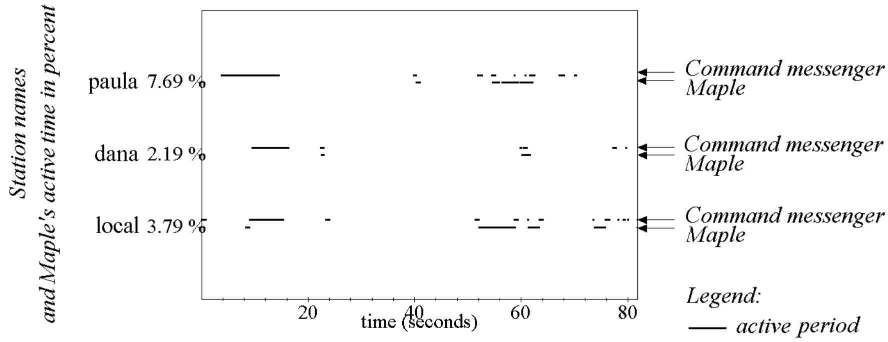

Fig. 3. Result of pvm[time] command from Example 1, Table 3

Example 2 from Table 3 presents a distributed variant of a square matrix-vector multiplication. A randomly generated 500-dimensional vector x, send to five different processors, is multiplied with local randomly generated 100×500 matrices, and the multiplication results are assembled into a final 500-dimensional vector z. The high usage percent reported in Figure 4.(a) as results of settime command indicate a possible efficient implementation of a distributed multi-

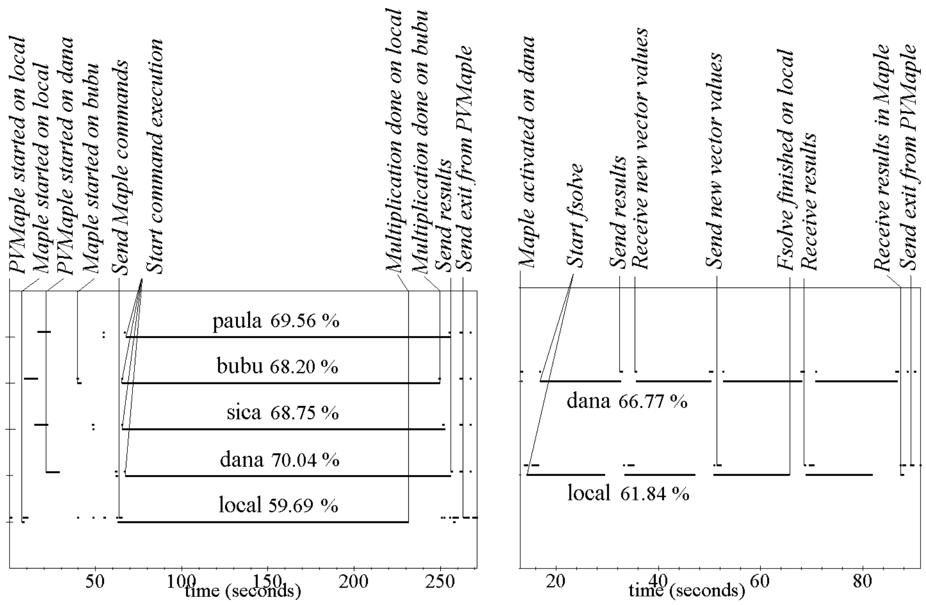

Fig. 4. Time diagrams and Maple's usage percent for (a) matrix-vector multiplication ($n \times n$, respectively n-dimensional objects, with $n = 500$) and PVMaple starting overhead (Example 2 from Table 3) (b) four steps of Hammer's method applied to an initial value problem of dimension $n = 40$ and PVMaple stopping overhead (Example 3)

plication algorithm. Indeed, comparing the time necessary to obtain the result of multiply(A, x) command with A a 500×500 matrix and the time depicted in Figure 4.(a) (between send Maple commands and send exit), we get a speed-up of 4.02 (using 5 processors).

The third sequence of commands presented in Table 3 is a part of a largest program for the numerical solution of initial value problems for ordinary differential equations of the form $y'(t) = f(t, y(t)),\ y(t_0) = y_0,\ f : [t_0, t_0+T]\times\mathrm{R}^n \to\mathrm{R}^n$. A short description of parallel solving strategies for such a problem is presented in [5]. One step of the particular Runge-Kutta method (iterative method), Hammer's parallel method [6], involves the solution of a $2n$-dimensional nonlinear system in the unknown n-dimensional vectors k_1 and k_2 on one processor and another $2n$-dimensional nonlinear system in the unknowns k_3 and k_4. Maple's function fsolve is used in this case. When n is large the processes inter-communications are insignificant relative to the time requested by fsolve calls. Figure 4.(b) indicates how the time is spent in applying four steps of the Hammer's method to a nonlinear initial value problem (convection-diffusion problem [6]) with $n = 40$ differential equations. The speed-up in this case is 1.59 (2 processors).

4 Proposed improvements and perspectives

PVMaple do not pretend to be better or more complex than the tools mentioned in the first section. It is designed for the common computer user which cannot afford to bye a parallel computer or another high-performance computer, but who has access to a workstation of a local network (a PC for example) with a licensed version of Maple, and who want to solve a problem requesting a large amount of computer resources. The system is intended to be a public domain tool (unlike ||Maple||). Based on PVM functions, it is faster than the system mentioned at the beginning which uses shared files for communications. Like Distributed Maple it provide an environment where parallel programming is possible within Maple. In Distributed Maple communications between the Java scheduler and the Maple processes are based on pipes; the corresponding Unix functions are not available under Microsoft Windows (a Maple V Release 4 problem). In PVMaple package currently available for PC's running Microsoft Windows a communication between Maple and its associated command messenger uses text files.

The PVMaple system is far from finished. It will be ported soon to Unix platforms. The command-messenger, which is written in standard C, must be recompiled when it is ported to a new operating system. The next step in our research is to extend the system introducing new functions with equivalents in the PVM library. We estimate that at the end of this summer, a beta-version of PVMaple will be available for freely download from author web page. Future implementation activities will follow two directions: experiments with applications written for PVMaple and command-messenger rewriting for interfaces with Mathematica and Matlab. Apart from extending PVMaple, our planes for future work include prototyping parallel algorithms for initial value problems (project D-NODE [6] based on Distributed Maple and also PVMaple).

5 Conclusions

This document described the basic concepts behind the Parallel Virtual Maple package for cooperative work of Maple processes on networks. We showed that by mapping Maple onto PVM we can get an efficient and extensible environment. This was realized through the use of PVM inter-communications which are handles to Maple inter-communications. The current prototyped system proved its usefulness from the software's point of view in solving large mathematical problems. Due to its relatively low communication/computation ratio it can be implemented in local networks. Large sets of small (PC class) workstations can be used as a virtual machine with quite high computational power.

Acknowledgments. The author would like to express her appreciation to Wolfgang Schreiner, the creator of Distributed Maple, and to thank him for the fruitful discussions and precious references.

References

1. Bernadin, L., Maple on a Massively Parallel, Distributed Memory Machine. In *PASCO '97: Second International Symposium on Parallel Symbolic Computation*, eds. M. Hitz, E. Kaltofen, ACM Press, New York (1997), 217-222.
2. Char, B.W., Progress Report on a System for General-Purpose Parallel Symbolic Algebraic Computation. In *ISSAC '90: International Symposium on Symbolic and Algebraic Computation*, ACM Press, New York (1990), 96-103.
3. Diaz, A., Kartofen, E., FoxBox: a System for Manipulating Symbolic Objects in Black Box Representation. In *ISSAC '98: International Symposium on Symbolic and Algebraic Computation*, ed. O. Gloor, ACM Press, New York (1998), 30-37.
4. Markus Fisher, PVM 3.4 port to Microsoft's Win32 Architecture, http://www.markus-fischer.de/win32project.htm.
5. Petcu, D. Implementation of Some Multiprocessor Algorithms for ODEs Using PVM. In *LNCS* **1332**: *Recent Advances in Parallel Machine and Message Passing Interface*, eds. M. Bubak, et al., Springer-Verlag, Berlin (1997), 375-382.
6. Petcu, D., Numerical Solution of ODEs with Distributed Maple, Technical Report 00-09 (2000), RISC-Linz, submitted to *Second Conference on Numerical Analysis and Applications*, Rousse, June 11-15 2000.
7. Schreiner, W., Distributed Maple – User and Reference Manual. Technical Report 98-05 (1998), RISC-Linz, and http://www.risc.uni-linz.ac.at/software/distmaple.
8. Schreiner, W., Developing a Distributed System for Algebraic Geometry. In *Euro-CM-Par '99: 3rd Euro-conference on Parallel and Distributed Computing for Computational Mechanics*, ed. B.Topping, Civil-Comp Press, Edinburgh (1999), 137-146.
9. Siegl, K., Parallelizing Algorithms for Symbolic Computation Using ||Maple||. In *4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ACM Press, San Diego (1993), 179-186.
10. Wang, D., On the Parallelization of Characteristic-Set-Based Algorithms. In *LNCS* **591**: *Parallel Computation - 1st International ACPC Conference*, ed. H.P. Zima, Springer-Verlag, Berlin (1991), 338-349.