Working with multiple Maple kernels connected by Distributed Maple or PVMaple

Dana Petcu
E-mail: petcu@info.uvt.ro

Abstract

This is a short note concerning the comparative structures and performances of Distributed Maple and PVMaple on a Unix cluster of SGI Octanes.[1]

1 Introduction

Message passing programming makes the task to solve a complex problem easier and faster for user. In particular, PVM and Java allows the user to wrote his application as a collection of cooperative tasks. On other hand, computer algebra systems like Maple can be successfully used in prototyping sequential algorithms for symbolic and numerical solution of mathematical problems. Therefore, constructing prototypes for parallel algorithms in Maple using the message passing paradigm is a challenging problem.

Parallel Virtual Maple (PVMaple) was created following the ideas of the previously build-up tool, Distributed Maple. Therefore there are many similarities between these two distributed computing environments.

The similarities and the differences between the two environments for coupling Maple kernels on clusters and network of workstations are discussed here. Section 2 describes the building concepts, while Section 3 presents more details about the tool structures. Section 4 treats the user interface, while Section 5 enters in more details about what is behind the user interface. Section 6 discuss some tests performed on a cluster of SGI Octanes. Based on these tests, Section 7 makes some recommendations concerning the usage of both distributed computing environments.

2 Concepts

Distributed Maple and PVMaple are extensions of Maple capabilities to distributed computations for workstations grouped into a parallel virtual machine, providing also facilities for post-execution analysis of the behaviour of a distributed computing session.

Distributed Maple's aim is to allow Maple applications to inter-operate across multiple heterogeneous hosts; it couples Maple kernels via Java. PVMaple's aim is to interface the flexible process and virtual machine control from PVM system with several Maple processes, allowing also the cooperation between Maple processes.

The main goals of both tools are the following ones:

[1]Elaborated during the author's research stage at RISC Linz, January 2001.

– to be public domain tools designed for anyone who want to solve a problem requiring a large amount of computer resources (http://www.risc.uni-linz.ac.at/software/distmaple, respectively http://www.info.uvt.ro/˜petcu/pvmaple);
– to be faster than previous designed systems;
– to allow parallel programming within Maple (the user interacts with the system via the text oriented Maple front-end)

Previous successful attempts to couple Maple kernels are presented in Table 1. Other CAS with distributed and parallel computing facilities are SciLab and MultiMatlab.

Table 1: Maple with parallel/distributed computation features

	Author	Year	Software	Specific
1	Char	'90	Sugarbush	Parallelism of C/Linda with Maple
2	Wang	'91	Maple kernels on different machines	Communication based on reading/ writing on shared files in global network file systems
3	Siegl	'93	||Maple||	Interface Strand – Maple
4	Bernardin	'97	Maple kernel for Paragon	Dedicated to parallel distributed memory machines
5	Diaz & Kartofen	'98	FoxBox	MPI mechanism allowing parallel/ distributed execution of FoxBox programs with interface to Maple

Note that DPToolbox (anson.ucdavis.edu/˜bsmoyers/parallel.html) is also very close to PVMaple's idea to maintain the PVM-function syntax into the user interface within the CAS. It realizes a multi-instance approach to support a convenient development of distributed and parallel Matlab applications. It uses PVM for process communication and control. Matlab instances are coupled via DPToolbox on heterogeneous computer clusters and parallel computers. The tool has three components: a Matlab/PVM interface that provides the communication primitives of the PVM system in Matlab, an interface for developing distributed and parallel Matlab applications, and an interface to manage parallel Matlab machines. It runs on Unix and Win32 platforms.

3 Structure

Distributed Maple and PVMaple enable distributed programming within Maple. The applications cannot be started in the absence of the public domain software which support the scheduler (Java, respectively PVM).

Distributed Maple is composed of two main components:

(a) a component which implements the scheduling mechanism for distributed applications and support for process inter-communication, and
(b) a binding between Maple and the scheduler.

The scheduler is PVMaple's case is the PVM daemon lying on each machine which participates to the distributed computing. The binding between the Maple process and the daemon is done by a special binary, namely command-messenger. Table 2 presents more details about the tools structure.

Table 2: Distributed Maple and PVMaple components and dependencies:

Component type	Distributed Maple		PVMaple	
	Component	Format	Component	Format
CAS	Maple V 4.0-6.0	binary	Maple V 4.0-6.0	binary
Communication	java	binary	pvmd	binary
CAS library	dist.maple	Maple text	pvm.txt/pvm.m	Maple text/internal
Binding	dist	Java classes	commandm	binary
Paths	dist.systems	text	paths.txt	text

4 User interface

4.1 Process and communication control

Typical functions for message passing paradigm are provided inside Maple: process creation, send and receive primitives, procedure for stopping the parallel virtual computing machine. Table 3 presents such functions.

Table 3: Process control and inter-communications via Maple function library dist.maple and pvm.m

Meaning	Distributed Maple	PVMaple
Create local or remote Maple processes	initialize	spawn
Send commands to Maple processes	start	send
Receive results from Maple processes	wait	receive
Stop distributed computing	terminate	exit

4.2 Non-equivalent commands

The syntax and functionality of the Maple commands presented in Table 3 are different.

Distributed Maple requests some knowledge of the computer type on which the Maple process will be started. This information is necessary to identify the binary paths. Multiple tasks on the same computer can be started by repeating the name of the computer in the starting list. PVMaple can start more than one Maple process on a machine by typing the number of tasks after the computer name; finding the binary paths is the responsibility of the command-messenger and the scheduler.

In Distributed Maple the user do not need to know where the Maple command will be executed: the send and receive commands do not make any reference to the machine where the commands are interpreted. Therefore the user must be careful only to start an appropriate number of Maple kernels. In PVMaple the send and receive commands can specify the exact process which will receive or send commands or data. The user has more control on what and where the computation will be done. Such a strategy is good when an heterogeneous computing environment is used or the participating computers have different overloads, but can be also a drawback since the user must have some knowledge about the status of the current distributed computing environment.

Distributed Maple allows only one Maple commands to be send at one moment. In order to reduce the network traffic, the send and receive command in PVMaple have been constructed in such a manner to allow multiple commands to be send once to all or a specific remote task. Therefore the syntax is more complicated than in Distributed Maple; the user must write a string of commands (by concatenation, for example). Moreover, using string of commands, the described Maple expressions will be remotely evaluated; writing the same expression in a start command from Distributed Maple, it will be evaluated by the local Maple process.

A second application of the fact that string of commands are used by send/receive functions is that the remotely Maple kernels (slaves of the master Maple process which has started them) can cooperate between them by sending and receiving data or commands. Exchanging information between Maple slave kernels is done in Distributed Maple using a virtual shared memory (data, get, put and clear commands).

Table 4 presents the distinct commands. Note that PVMaple has no possibility to simulate shared data objects like Distributed Maple. On other hand, PVMaple has a better control on the task (task identifying number in the entire system, machine on which it's running, task identifying number on a machine).

Table 4: Commands without equivalence

Purpose	Distributed Maple	PVMaple
Visualization	trace, label, visualise	settime, time
Virtual shared memory	data, get, put, clear	-
Components identification	-	ProcID, MachID, TaskId, Tasks

Concerning the time diagrams, Distributed Maple offers more facilities than PVMaple. It allows not only three kinds of post-processing time diagrams, but also online-visualization (both are not using Maple visualization capabilities, but Java's ones). Export of diagrams is possible using some external software. PVMaple allows only post-visualization of a session evolution using the Maple plot function (the user do not need to exit from Maple to obtain a time diagram). Export of diagrams is possible inside Maple. The selection of the desired time interval to be displayed is done in Distributed Maple using labels; in PVMaple, the user can specify the time interval (in Maple format) in the time function which plots the time diagram. The label option offers more precision, but a detail request implies new labels and rerun. The time interval option is user-dependent; further details do not need a new run, but new calls to the time plot function.

The time diagrams provided by Distributed Maple are more suggestive. Different information can be extracted from them. The only advantages of using PVMaple's time diagram are that it draws not only the Maple processes active time but also that of the companion software (command-messenger active time). Moreover the percent of the active time in a session of a Maple process is displayed on the time diagram after the name station on which the process has been run.

The active time periods are registered by PVMaple into some text file which remain in a temporary directory after the virtual machine has been stopped and can be used also by other programs in further time diagram processing. Distributed Maple uses also time registration in text files which are further interpreted by appealing to a Java visualisation class.

5 Behind the user interface

Distributed Maple is more easy to be used since one must just start Maple and type the commands (the scheduler and the binding components are started from inside Maple). PVMaple can run only after PVM and the command-messenger (daemon program which will wait for user commands) have been started outside Maple.

The communication between Maple and the binding component is done in Distributed Maple using pipes. In PVMaple the communication between Maple and command-messenger is done by using text files (this a drawback when the communicating data or commands are very large). This difference is due to the fact that PVMaple is available also for Win32 systems, on which Maple has no implementation for the process control routines. At this moment Distributed Maple is available for Unix systems (the main problem with the Distributed Maple implementation on Win32 systems is within the Maple interface).

The command-messenger appeals to PVM functions for process control and communications. Maple remote kernels and they companion command-messengers are started by PVMD as a result of PVM's spawn appeal. The command-messenger has also the functionality of registering the active time periods.

Distributed Maple is more fault tolerant than PVMaple. Error cases are carefully studied by the first tool. PVMaple is more sensitive: errors in launching remote processes and exchanging information are traced using PVM function and reported to the Maple user as in PVM, but errors in computations are reported sometimes partially (when a sequence of commands is send to a remote process only the result of the last command is send back since the command sequence is seen as a part of a Maple procedure – the return command can be used the user in the command sequence for special cases).

The documentation for Distributed Maple is more elaborated. Actually PVMaple is distributed with only a few description lines and a small number of examples. Supplementary information are provided for Distributed Maple in [7] and [8] and for PVMaple in [5] and [6].

6 Performances

Tests were performed on a cluster of three SGI Octanes. Three significant examples are given here.

The first example is simple: four integrals are computed in four distinct Maple process. Table 5 presents the code of the two implementations. PVMaple command syntax allows in this case to send one command to all tasks which will perform different computations, depending on the task identifying number; in Distributed Maple the commands must be send in a sequence (almost a sequential program). The time measurements are not relevant since the computation time (of miliseconds order) is very small relative to the time necessary to send the commands or to start the processes (around 5 seconds in both cases).

The second example is taken from the example directory of Distributed Maple: factorization of a list of long integers. Table 6 and 7 presents the implementation in both environ-

Table 5: Computing four integrals using 4 Maple kernels (3 remote, 1 local) running on 3 computers

```
> # Distributed Maple
> read `dist.maple5`; dist[trace](`integr`):
Distributed Maple V1.1.8 (c) 1998-2000
Wolfgang Schreiner (RISC-Linz)
See http://www.risc.uni-linz.ac.at/
software/distmaple
> dist[initializep]([[milkyway,octane],
> [pinwheel,octane],[whirlpool,octane],
> [pinwheel,octane]]);
connecting milkyway...
connecting pinwheel...
connecting whirlpool...
connecting pinwheel...
          okay
> for i from 1 to  4 do
>   s[i]:=dist[start](int,x^i,x): od:
>
>
> dist[wait](s[2]);
```

$$\frac{1}{3}\,x^3$$

```
> dist[terminate]();
          okay
```

```
> # PVMaple
> read `pvm.txt`;
PVMaple version 1.0
Temporary directory: /home/petcu/tmp/
(you can chage it;
example: `pvm/temp`:=`/temp/`)
> pvm[spawn]([pinwheel,1],[milkyway,2],
> [whirlpool,1]);
Try to contact pinwheel for 1 tasks
Try to contact milkyway for 2 tasks
Try to contact whirlpool for 1 tasks
Active tasks: [[`local`,1],[pinwheel,1],
[milkyway,1],[milkyway,2],
[`whirlpool.risc.uni-linz.ac.at`, 1]]
># for i from 1 to 4 do
>#  pvm[send](`pvm/tasks`[i],
>#       cat(`int(x^`,i,`,x)`)): od
> pvm[send](`all`,`int(x^pvm[TaskId],x);`);
> pvm[receive](`all`,[milkyway,1]);
```

$$\frac{1}{3}\,x^3$$

```
> pvm[exit]();
PVmaple quitted
```

ments. Similar execution times have been obtained with both tools, although in background the implementation are different: on Distributed Maple more commands are send, for each integer one Maple command (with different data), and in PVMaple each task receive a part of the initial list (the same command, the same data, but the execution depends on the process identifiers). The time reported by Distributed Maple is strictly for integer factorization is 9.88 s and by PVMaple 12.89 s, due to an imbalanced load between the two workers (in this case the task pool strategy of Distributed Maple is more efficient than the static distribution adopted by PVMaple). On other hand more time is spent by Distributed Maple to prepare the tasks start (on 10th second versus the 5th second in PVMaple). Moreover, recovering the results from the remote processes seems to be slower than in the case of PVMaple. In the PVMaple's time diagram we can see clearly that the master Maple process who has started the other two processes has apparently no job to do. Apparently only since one process has been started on the same machine on which the master Maple process is running. One second option is to give some job also to the master Maple process, as we have done in the third example.

The third example is a more complicated one: the goal is the integration of a initial value problem for ordinary differential equations with Hammer's parallel method of Runge-Kutta type (two Maple processes will be running). More details about the test problem and the numerical method used can be found in [4]; same paper reports previous tests done with Distributed Maple solving large initial value problems. Table 8 and 9 presents shortly

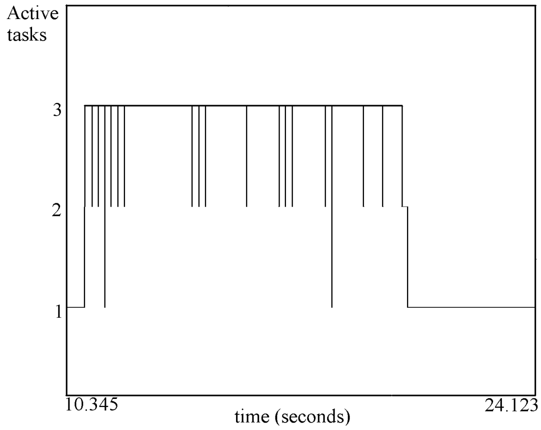
Table 6: Distributed integer factorizations using 2 Maple kernels running on 2 remote computers – Maple sources

```
> # Distributed Maple

> # file ifactor.maple
> # Distributed Maple v 1.7 1998/05/30: distributed integer factorizations
> # return a list of ‘length‘ elements each of which is a pair [i, f] where
> # ‘i‘ <= ‘size‘ is a random integer and ‘f‘ is the factorization of ‘i‘

> main := proc(length, size)
>   local numbers, results, ti;
>   dist[all](‘readlib(ifactors);‘);
>   numbers := rands(length, size);
>   results := dfactors(numbers);
>   [seq([numbers[i],results[i]],
>   i=1..length)]; end:
> dfactors := proc(numbers)
>   local length,tasks,results,i,t,r;
>   length := nops(numbers);
>   tasks := [];
>   for i from 1 to length do
>     t:=dist[start](ifactors,numbers[i]);
>     tasks := [op(tasks), t]; od;
>   results := [];
>   for i from 1 to length do
>     r := dist[wait](tasks[i]);
>     results := [op(results), r]; od;
>   RETURN(results); end:
> rands := proc(length, size)
>   local r, i, n, numbers;
>   r := rand(size);
>   numbers := [];
>   for i from 1 to length do
>     n := r();
>     numbers := [op(numbers), n]; od;
>   numbers; end:

> # Main program
> read ‘dist.maple5‘;
Distributed Maple V1.1.8 ...
> dist[trace](‘intregi‘):
> dist[initialize]([[pinwheel,octane],
>  [whirlpool,octane]]):
connecting pinwheel...
connecting whirlpool...
       okay
>
> dist[label](‘exec‘):
> read ‘ifactor.maple‘:
> main(20,2^90):
> dist[terminate]();
           okay
```

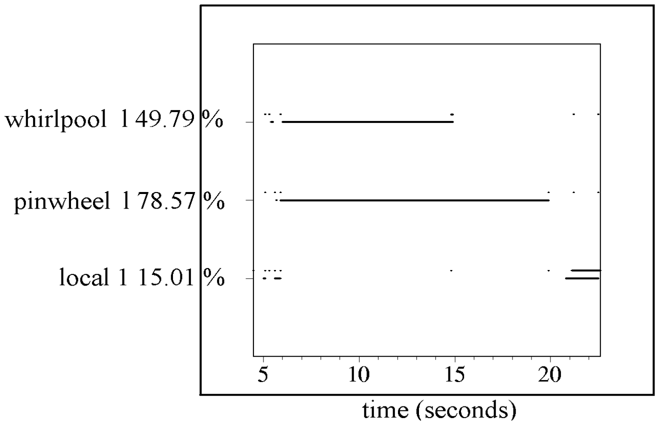
```
> # PVMaple

> main := proc(length, size)
>  local numbers, results, ti;
>  # pvm[send](‘all‘,‘readlib(ifactors);‘);
>  numbers := rands(length, size);
>  results := dfactors(numbers);
>  [seq([numbers[i] results[i]],
>  i=1..length)]; end:
> dfactors := proc(numbers)
>  local length,tasks,results,i,t,r;
>  length := nops(numbers);
>  t:=pvm[send](‘all‘,cat(‘readlib(ifactors):
>   numbers:=‘, convert(numbers, string),
>   ‘: s:=[]; for i from 1 to nops(numbers)
>   do if(i mod ‘,convert(No_mach,string),‘)‘,
>   ‘=pvm[TaskId]-1 then s:=[op(s), ifactors
>   (numbers[i])]; fi; od: s;‘));
>
>  results:=pvm[receive](t,‘all‘);
>  RETURN(results); end;
> rands := proc(length, size)
>  local r, i, n, numbers;
>  r := rand(size);
>  numbers := [];
>  for i from 1 to length do
>    n := r();
>    numbers := [op(numbers), n]; od;
>  numbers; end:

> read ‘pvm.m‘;
>
>
> pvm[spawn]([pinwheel,1],
>   [whirlpool,1]);
Try to contact pinwheel for 1 tasks
Try to contact whirlpool for 1 tasks
Active tasks: [[‘local‘,1], [pinwheel,1],
[whirlpool,1]]
> No_mach:=nops(‘pvm/tasks‘)-1:
> read ‘ifactor.maple‘:
> main(20,2^90):
> pvm[exit]();
PVmaple quitted
```

Table 7: Distributed integer factorizations – graphical results

```
> # Distributed Maple
> # outside Maple: java visual.Main intregi
> # exec terminate 100 100
```




```
> # PVMaple,  inside Maple:
> # after pvm[time](‘all‘)
> pvm[time](4.8..22.4);
```



the Maple codes and time diagrams for two integration steps. For one integration steps (normally there are hundreds) PVMaple needs twice more time than Distributed Maple. The difference is significant. This is a consequence of the above mentioned fact that the Maple implementation on Win32 do not allow process control, and therefore, to use the same code on Win32 and Unix, the communication between a command messenger and a Maple process is performed by reading/writing some files and not by pipes like in Distributed Maple. Consequently the communication is much slower when the exchanged messages are long.

7 Recommendations

Distributed Maple is recommended when the information exchanged between Maple kernels is large or in the case when the shared memory model of computation is requested by the problem to be solved. PVMaple is recommended when the exchanged Maple commands and data are relative small, or when the problem solving algorithm requests the information exchange between remote Maple kernels, or when an entire list of Maple commands must be send at once to a remote Maple process.

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.

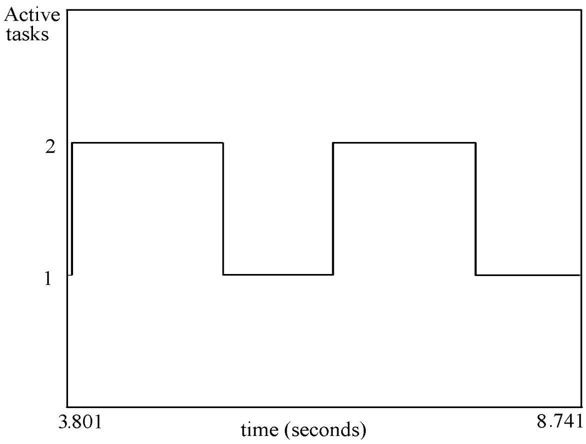
Table 8: ODE numerical solver– Maple sources for problem and method

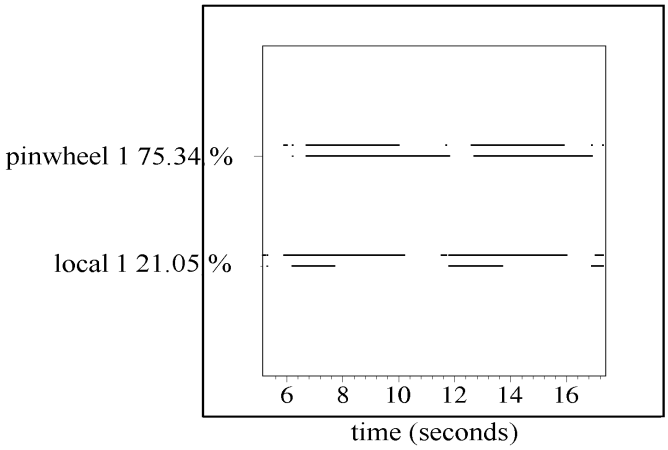
```
> # file IVP.txt

> # IVP with exact solution y[i]=1/t
> initial_values:=proc() global n; local i; vector(n,[seq(100,i=1..n)]); end:
> right_side_system:=proc(t,y) local i,n; n:=vectdim(y);
>   vector(n, [-y[1], seq(499999./(n-1)* sum('y[k]', 'k'=1..i-1)-((i-1)*
>   499999./(n-1)+1)*y[i], i=2..n)]) end:
> interval:=proc() [0.01,1.] end:
>
> # Integration procedure
> integration:=proc(method,h) local y,i,t; global n,no_steps;
> y:=initial_values(); i:=1: t:=interval()[1]+h;
> while t<=interval()[2] and i<=no_steps do
>      y:=method(y,h,t); print(t,y[1]); t:=t+h; i:=i+1: od; end:
>

> #Distributed Maple                        >#PVMaple

> # file method.txt

> # Use: Hammer's numerical method
> # Distributed version
> Hammer_dist:=proc(y,h,t) local K12,i,S,R,xx,K1,K2,K3,K4; global k1,k2,k3,k4;
>   k1:='k1': k2:='k2': k1:=vector(vectdim(y)): k2:=vector(vectdim(y)):
>   xx:=convert(evalm(k1-right_side_system(t+h*(3-sqrt(3.))/6.,  evalm(y+h*(5*k1+
>      (1-2*sqrt(3.))*k2)/12.))), set) union convert(evalm(k2-right_side_system(t+
>      h*(3+sqrt(3.))/6., evalm(y+h*((1+2*sqrt(3.))*k1+5*k2)/12.))), set),
>      convert(evalm(k1),set) union convert(evalm(k2),set):

>   S:=dist[start](fsolve,xx[1],xx[2]);   > S:=pvm[send]([machine,1],cat(`ww:=fsolve(`,
>                                         > convert(xx[1],string),`, `,
>                                         > convert(xx[2],string),
>                                         > `): assign(ww): [op(convert(k1,list)),
>                                         > op(convert(k2,list))];`)):

>   k3:='k3': k4:='k4': k3:=vector(vectdim(y)): k4:=vector(vectdim(y)):
>   R:=fsolve(convert(evalm(k3-right_side_system(t+h*(3-sqrt(3.))/6,evalm( y+h*
>      (1/2*k3-sqrt(3.)*k4/6)))),set) union convert(evalm(k4-right_side_system(t+
>      h*(3+sqrt(3.))/6, evalm(y+h*(sqrt(3.)*k3/6+k4/2)))),set),convert(evalm(k3),
>      set) union convert(evalm(k4),set)); assign(R):

>   K12:=dist[wait](S); assign(K12);      > K12:=pvm[receive](S,[machine,1]);
>                                         > k1:=[seq(K12[i],i=1..n)];
>                                         > k2:=[seq(K12[i],i=n+1..2*n)];
>   K1:=convert(k1,vector): K2:=convert(k2,vector): K3:=convert(k3,vector):
>   K4:=convert(k4,vector): evalm(y+h*(3*K1+3*K2-2*K3-2*K4)/2); end:
```

Table 9: ODE numerical solver– testing Maple code and graphical results

```
> #Distributed Maple
> read `dist.maple5`;
Distributed Maple V1.1.8 ...
> dist[trace](`Q-Hammer`):
> dist[initialize]([[pinwheel,octane]]):
connecting pinwheel...
       okay
> n:=20: no_steps:=2:
> dist[label](`exec`):
> read `ivp.txt`; read `method.txt`;
> integration(Hammer_dist,0.000001):
      .010001, 99.999990000
      .010002, 99.999980000
> dist[label](`fin`): dist[terminate]();
           okay
>#outside Maple: java visual.Main Q-Hammer
>#exec fin 100 100
```

```
>#PVMaple
> read `pvm.m`;
>
> machine:=`pinwheel`:
> pvm[spawn]([machine,1]);
Try to contact pinwheel for 1 tasks
Active tasks: [[`local`,1], [pinwheel,1]]
> n:=20: no_steps:=2:
>
> read `ivp.txt`; read `method.txt`;
> integration(Hammer_dist,0.000001):
        .010001, 99.999990000
        .010002, 99.999980000
> pvm[exit]();
PVmaple quitted
> #inside Maple: after pvm[time](`all`);
> pvm[time](7.1..17.3);
```





[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] Petcu, D., Numerical Solution of ODEs with Distributed Maple, Technical Report 00-09 (2000), RISC-Linz, to appear in LNCS series: *Second Conference on Numerical Analysis and Applications*, Rousse, June 11-15 2000.

[5] Petcu, D., PVMaple: a Distributed Approach to Cooperative Work of Maple Processes, In *LNCS* **1908***: 7th European PVM/MPI User's Group Meeting, Balatonfüred, Sept. 2000*, ed. J. Dongarra, P. Kacsuk, N. Podhorski, 216-224

[6] Petcu, D., A Networked Environment for Solving Large Mathematical Problems, submitted to EuroPar'01, Manchester, sept. 2001.

[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.