Comparison of Parallel Simulation Techniques
Workstation Cluster / MATLAB / PSI

MATLAB is a general tool for mathematical and engineering calculations and visualisations. It is very well known and widely in use. Therefore a comprehensive introduction is not necessary. Although MATLAB is not a special simulation tool it is often used for small and medium simulation problems. In spite of its excellent features there are some lacking capabilities. One is distributed and parallel processing. In similar tools like Matrixx and CTRL-C we find the same lack, too. In order to overcome this lack some investigations were done at the University of Rostock and at the Fachhochschule Wismar. By the authors the C++ class library PSILIB for transport independent interprocess communication and process control between heterogeneous platforms was developed. This library considers not only standard UNIX derivatives, but also real time operating systems like OS9 and LynxOS. On top of this library common engineering standard tools with a C interface can be extended to programming environments for distributed and parallel applications. MATLAB performs this with its MEX interface. Today, there are of course a number of communication libraries with wider distribution, which can be used in the same way. A much more serious problem is the construction of a high level and easy to use interface within MATLAB to access a communication library via the MEX interface. We examined the two logical approaches shared and distributed memory. Physically both are implemented by message passing. For solving the tasks of this comparison we used the "Distributed Memory Interface". To give an impression of the interface handling the M-Code for the Monte-Carlo study is discussed in greater detail.

All test examples have been run on a cluster of 20 SUN Classic workstations under Solaris 2 connected via Ethernet (10 Mbit/s). Static load balancing was used exclusively.

a) Monte-Carlo Simulation: Only a task division into subtasks producing equal load is useful. That means for 1000 parameter variations 2, 4, 5, 8, 10, ... subtask are suitable. As can be seen in the table below, the resulting speed-up factor f grows almost linear with the number of subtasks M.

M	2	4	5	8	10
f	2.00	3.99	4.99	7.96	9.92

Due to the possible mixture of programming and interactive/interpretative execution in MATLAB the necessary time expenditure for implementing and testing a problem solution is very small compared with compilation based approaches like C or FORTRAN programming. The supported matrix oriented notation leads to short and compact code, how the example M-Function `ex1()` for the Monte-Carlo study shows.

```
function [xmean]=ex1(dd)
   t0=0; tf=2; h=0.001; x0=[0 0.1]';
   global d
   xmean=zeros((tf-t0)/h+1,1);
   for i=1:length(dd)
      x=rk4('mass_spring',t0,tf,h,x0);
      xmean=xmean+x(:,2)/length(dd);
   end
return
```

The experiment is carried out by simply typing the following lines.

```
dd=800+400*rand(1000,1);
x=ex1(dd);
plot(x);
```

As a first attempt we can try a parallel solution in the same manner. After starting `nslaves` MATLAB instances and generating damping factors, one column of the random matrix is put to each slave.

```
nslaves=10;
slaves=spawn(nslaves);
dd=800+400*rand(1000/nslaves,nslaves);
for i=1:nslaves
   put(slaves(i),dd(:,i))
end
```

Now each slave can settle its part of the whole task and the interactive experiment is finished by putting back the results from the slaves and calculating the average response.

```
aeval(slaves,'x=ex1(dd);')
x=0;
for i=1:nslaves
   x=x+putback(slaves(i),'x')/nslaves;
end
plot(x);
```

The experiment above is done in a parallel fashion without any programming! From such a successful test it is a close step to a real master/slave program.

Master M-File:

```
nslaves=10;
slaves=spawn(nslaves,'ex1_slave');
dd=800+400*rand(1000/nslaves,nslaves);
for i=1:nslaves
   put(slaves(i),dd(:,i))
end
x=0;
for i=1:nslaves
   x=x+get(slaves(i))/nslaves;
end
plot(x)
```

Slave M-File:

```
dd=get;
put(-1,ex1(dd))
```

Sometimes the SPMD (single program multiple data) paradigm is preferred. Although it is not particularly suitable in this example, because the problem is logically structured in a master/slave manner, it saves one processor for the same degree of parallelism.

SPMD M-File:

```
n=10;

if parent
  ids=myid;
  ids(2:n)=spawn(n-1,'ex1_spmd');
put(ids(2:n),ids))
dd=800+400*rand(1000/n,n);
  for i=2:n
    put(ids(i),dd(:,i)
  end
  dd=dd(:,1)
else %child
  ids=get;
  dd=get;
end

x=ex1(dd)/n;

if parent
 for i=2:n
   x=x+get(ids(i));
 end
  plot(x)
else %child
 put(ids(1),x)
end
```

b) Coupled predator-prey population: The SPMD paradigm was used and all five tasks were located on different processors. The communication interval was varied from c_{int}=h to c_{int}=20h.

c_{int}	h	2h	5h	10h	20h
f	0.70	0.98	1.90	2.77	3.71

Speed-up factors greater than one are not reached until c_{int} is greater than 2h.

c) Partial differential equation: The task was solved with a discretisation into N=800 lines using the SPMD paradigm. At first the number of parallel tasks M was varied from 2 to 16.

M	2	4	5	8	10	16
f	1.83	3.07	3.60	5.01	5.46	7.11

Then the communication interval was increased for M=8 tasks.

c_{int}	h	2h	4h	6h	8h
f	5.01	5.71	6.24	6.48	6.54

One reason for the relatively high speed-up factors compared with other solutions in this series is the slow implementation of the RK4 algorithm. Because there is no RK4 with fixed stepsize in MATLAB, it was implemented for the comparison in M-Code. The time needed by an experienced MATLAB user for implementing and testing the parallel versions with the "Distributed Memory Interface" is nearly the same as for the serial solution. Principles of the "Shared Memory Interface" and performance tests of both interfaces using PVM as alternative communication library will be published in the near future.

S. Pawletta, W. Drewelow, Inst. of Automatic Control, University of Rostock, D-18051 Rostock;
T. Pawletta, Chair of Applied Computer Science, Fachhochschule Wismar, D- 23968 Wismar
Email: sven.pawletta@etechnik.uni-rostock.de