C9 Fuzzy Control of a Two Tank System, **SNE 17**, (7/96), asks for approaches and for implementations of modules for fuzzy control.
SP: support for fuzzy control, two-dimensional calculations for control surface, pure discrete approach possible

C10 Dining Philosophers II, **SNE 18** (11/96), reviews discrete simulators with respect to concurrent access to resources and with deadlocks.
SP: discrete random variables, simultaneous events, deadlock recognition

C11 SCARA Robot, **SNE 22** (3/98), deals with implicit and hybrid systems with state events.
SP: implicit model, different approaches for collision event and action

C12 Collision of Spheres, **SNE 27**, November 1999, allows numerical or analytical analysis as well as continuous or discrete approaches
SP: broad variety of approaches (numerical - continuous, numerical – discrete, numerical – analytical, analytical – symbolic), collision limit

C13 Crane Crab with Embedded Control, **SNE 31** (3/01), revised **SNE 35/36** (11/02) checks techniques and features for embedded digital control with sensors and with DAE-systems
SP: implicit model, discrete control coupled with sensor diagnosis, complex experiments

C14 Supply Chain, **SNE 32/33** (11/2001), **SNE 34** (7/2002) addresses discrete simulators - features for supply chain systems (messages, strategies)
SP: distinction between material flow and order flow, distance-dependent control strategies

C15 Clearance Identification, **SNE 35/36** (11/02), checks identification features (based on measured data) and influences of noise
SP: identification algorithms, short-term input functions (Dirac-like), support of statistics

Solutions

We invite all readers to participate in these comparisons. Please, simulate the model(s) with any tool of your choice and send in a solution. A solution should consist of: a short description of the simulator, modelling technique, model description, and results of the three tasks. Additionally we ask for model sources

The solution should fit into one page of **SNE** – templates are found at our web page. Solutions sent in are reviewed.

Felix Breitenecker
Felix.Breitenecker@tuwien.ac.at

A MATLAB – based Solution to ARGESIM “Comparison of Parallel Simulation Techniques” using a DP-Toolbox

R. Fink, S. Pawletta, T. Pawletta, University of Applied Science Wismar, Germany
r.fink@stud.hs-wismar.de

Simulator: MATLAB is a widely used tool for rapid prototyping in engineering sectors. One big disadvantage of MATLAB is the lack of parallel processing features. To overcome this disadvantage, several tools were developed expanding MATLAB by those features. One of these tools is the *DP-Toolbox*, where DP stands for distributed and parallel.

The *DP-Toolbox* was developed from 1995 until 1999 at the University of Rostock and since 2002 at the University of Applied Science Wismar. Communication and synchronisation is done via message passing using the software package PVM (Parallel Virtual Machine). The DP-Toolbox consists of two levels: DP-High and DP-Low. At the DP-Low level, PVM functions were mapped to MATLAB functions using MATLAB’s MEX interface. At the DP-High level, low level functions will be bundled into easy to use high level functions. The toolbox was mainly developed for Unix platforms, where older versions (1.4.x) also support Win32 systems.

Task a: Monte Carlo study: All simulation runs can be performed independently. Therefore, the basic idea is to distribute the runs to the available processors. Since the single simulation runs took similar times, static load balancing was used. For balancing the load statically only one DP-High function (`dpscatter`) was needed. Due to the powerful high level functions, the master program is very short:

```
function [t,xmean]=mcs_master(num_slaves);
……….
% parameters
num_runs   = 1000;
t0 = 0; tf = 2; h  = 0.001;
t=t0:h:tf; % returned time values
x0 = [0;0.1]; % initial conditions

% generate damping factors
d = 800 + 400 * rand(num_runs,1);

% start up slaves
slaves=dpspawn(num_slaves,Try,mcs_slave);

% distribute damping factors
% and simulation parameters
dpscatter(slaves,d);
dpsend(slaves,[t0,tf,h,x0']);
% gather results
xmean = mean(dpgather(slaves));
```

The whole code for the slave program is even shorter:

```
function mcs_slave()
…………
% receive damping factors and simulation
% parameters
d   = dprecv; par = dprecv;
t0=par(1); tf=par(2);h=par(3);x0=par(4:5)';

% perform partial monte carlo study
global D;
xmean = 0;
for i = 1:length(d)
      D = d(i);
      x = rk4('mcs_equ',t0,tf,x0,h);
      xmean = xmean + x(:,2) / length(d);
end

% send result to master
dpsend(dpparent,xmean);
```

After all, the implementation of the parallel program took a few more time than the implementation of the serial program.

For all calculations, the RK4-algorithm with an integration stepsize of *h=0.001* was used. The table below shows the speedup factor *f* for a different number of processors *M*.

M	**2**	**4**	**6**	**8**
f	1.86	3.42	4.65	5.62

The reasons of the relatively low speedup values are the start-up times of the slave processes. Because for every slave one MATLAB instance has to be started, which takes up to 10 seconds to start up, this factor must be considered in small sized problems.

Task b: coupled predator-prey population. The five populations were distributed to five slave processes controlled by a master process. The master itself starts the slaves, sends the initial values towards them and finally collects the results.

Calculations were proceeded with the RK4-algorithm using an integration stepsize of *h=0.01*. The following table shows the speedup factor depending on different communication intervals *cint.*

cint	***h***	***2h***	***5h***	***10h***	***20h***
f	0.13	0.22	0.37	0.48	0.56

As the table shows, communication was not successful. The reason for these negative results is the tight coupling between the differential equations leading to relatively high communication times.

Task c: partial differential equation: The parallelization of this example was also realized by a master-slave program.

As in task b, the master program starts up the slaves, distributes the parameters and collects the results from them. The slave processes exchange only the boundary values of their simulated areas.

The following table shows the speedup factor *f* for a different number of processors. RK4-algorithm with stepsize *h=0.005* and a number of lines *N=800* was used for all calculations.

M	**2**	**4**	**6**	**8**
f	0.29	0.2	0.16	0.15

The table shows, that at increasing processor numbers speedup values will decrease. The reason for this effect is the increasing number of simulated areas and therefore the rise of boundaries where simulation values will be exchanged. This leads to a rising communication expense which can't be compensated by the decreasing calculation expense.

Conclusions: The examples show that parallelization using a cluster of workstations is not suitable for every type of simulation. If there are more than one simulation runs running independently, parallel processing can be successful and can be implemented easily. On the other hand, if only one simulation run must be done, parallelization is harder to implement and conditionally successful.

To compare implementation times, the described examples have been realized in C/PVM also. These implementations took approximately four times longer than developing under MATLAB/DP-Toolbox.

Hardware: All examples have been run on a cluster of 8 workstations containing a 1500 MHz AMD processor. The computers were coupled by a switched Gigabit Ethernet.

Software: Linux operating system with the newest version 1.5.0 DP-Toolbox (newest version) and MATLAB (6.5).

.