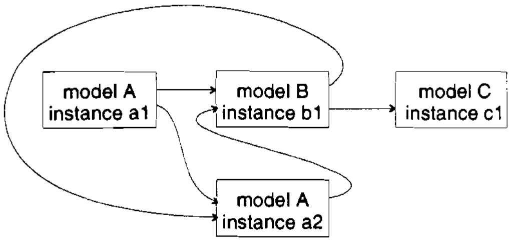
Comparison of Parallel Simulation Techniques
Cogent XTM / "mosis"

"mosis" (modular simulation system) is a CSSL-simulation system specially designed for submodel structures and mapping them on multiprocessor systems with distributed memory, like transputer systems, workstation clusters etc. It has been developed at the Dept. Simlation Technques of TU Vienna on the basis of "C" and has most features of other simulation languages based on differential equations (various integration algorithms, event handling) with the possibility to expand the system by user functions. It is designed to work on many different hardware platforms, not only on parallel, but also on serial machines like single workstations and PC's (multitasking!) where the user interface (except from windowing etc.) is everywhere the same independent from the hardware. It can also be used (without the parallel features) as an all-purpose simulation language.

"mosis" consists of a "mosis"-to-"C" compiler and a run-time system in object code to link user models. The run-time system contains I/O routines, graphics, a "C"-like interpreter language and an object oriented simulation kernel. For a simulation study, at run-time first instances of the models (submodels) must be generated (containing all the data, allocated at run-time) and connected together, if necessary (see figure); then the simulation can be started (before that the user can set parameter values, etc). This concepts allows instancing and simulating simultaneously different models.



In "mosis" it is possible to split a bigger model into several smaller models that can be simulated on different processors (and connect them via special unidirectional "mosis"-links; see figure). The connections have to be done at run-time (without having to re-compile the whole system). At each communication interval the data of the output values in one instance are sent to the corresponding input signals of the other instance.

Models are compiled to "C"-files and linked to the run-time system. At this level, the user can access models ("A", "B") in the following way:

```
int a1; int b1;
a1=instance("A",2); b1=instance ("B",1);
```

The second parameter tells the system on which processor the instance should be created (on parallel computer systems). Two instances are linked together with the "connect" command. When one instance is started, all other instances depending on this instance, are also started automatically. The values of a variable depending on the time are saved with the "watch" command:

```
connect(a1.oval1,b1.i);
watch(a1.x); run(a1);
```

All simulations are done in the background (also on PC's); during the calculation it is possible to view values of variables continuously, to stop or start instances or to enter any other command.

"mosis" has been currently implemented on the following systems: PC's (Borland C), 386+ (Watcom C/32), UNIX workstations with X Window and PVM, and the transputer workstation Cogent XTM ("mosis" version 1.0 α). "mosis" is free software; it may be copied for non-commercial use and will be available with the first distributed test version 1.0 β on the simulation server "simserv.tuwien.ac.at" via anonymous ftp (from Sept.'94 on).

The Cogent XTM consists of 20 Transputers T800 with 20MBit-Links and a faster bus connection for short messages. Each transputer has 4 MB of local memory (without swapping). For communication, "Kernel Linda" is used which simulates some kind of a shared memory pool for all processes. The operating system is QIX. The system is dynamically scalable.

All simulations within "mosis" have been done using double precision (standard); the Kernel Linda calls are used to simulate a message passing mechanism between the tasks. "mosis" and all comparison examples were compiled using the same ANSI-C compiler on the Cogent. As requested, in all of the comparisons the RK4 algorithm was used (different stepsizes).

a) Monte-Carlo Simulation: This problem was solved with 8 transputers with a static load balancing: Every transputer had to simulate 125 times the system and store the simulation results. In the last run, the sum of all simulated values is sent to the main process. In this case, this is the best solution, as all processors have the same speed and all are exclusively used by this calculation. Therefore all processes need nearly the same time to simulate their tasks (speed variation < 0.5 %).

Although the conditions were nearly ideal for this comparison, the resulting speed-up factor for eight processors was only f=4.4. The reason of this relatively bad value is probably that the polling for received signals during the simulation run takes much time. In the first distributed version of "mosis" this overhead will be significantly reduced.

Model description:

```
model mcarlo()
{ /* parameter declarations etc. */
  initial { /* initialisation  */ }
  dynamic { derivative {
        x'=xp,0.1;
       xp'=(-dgauss*xp-kx*m)/m,0.0;
    } } }
```

Commands at run-time :

```
int i; int p[8];
for(i=0;i;i++) p[i]=instance("mcarlo",i+1);
/* on processor i+1 */
for(i=0;i;i++) run(p[i]);
/* start all 8 instances simultaneously */
```

b) Coupled predator-prey population: Parallelisation of this example was not successful (because of calculation speed >> communication speed). First, with communication performed at every integration step (cint = h = 0.01, RK4), the parallelisation resulted in a "speed-up"-factor of f=0.06; with reduced communication (cint=10h) f increased to f=0.3 (3.3 times slower). Increasing the communication interval to cint=20h led to a factor of f=0.61 (1.61 times slower). All five tasks were located on different processors, therefore the communication overhead was very high.

c) Partial differential equation: This example was simulated with a discretisation into N=800 lines; 8 processors were used here, each of them held 100 lines or 200 state variables; at each communication point the boundary values were interchanged between neighbouring blocks (8 tasks; 14 connections).

Experiments were done by varying the communication interval from cint=h (0.005) to cint=8h; the resulting speed-up factors (results with cint>8h unstable):

cint	h	2h	4h	8h
f	4.33	4.56	5.64	6.04

The time used for developing the parallel versions was the same as for the serial solution, as they could easily transformed into one another and the message passing algorithms are hidden to the user: he/she can use the I/O signals like common variables; input and output is automatically done by the simulation system.

G. Schuster, F. Breitenecker, Dept. Simulation Techniques, Technical University Vienna, Wiedner Hauptstraße 8-10, A-1040 Wien, Austria.