Comparison of Parallel Simulation Techniques Cogent XTM / SIMUL_R PARALLEL

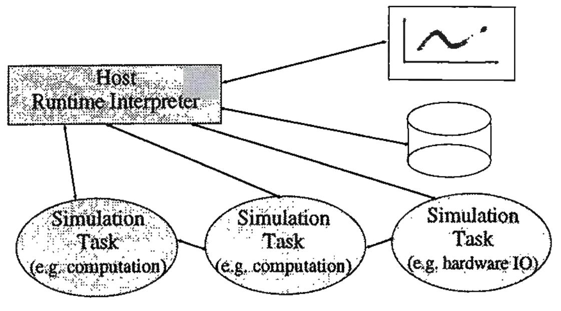
Host
Runtime Interpreter
Simulation Task (e.g. computation)
Simulation Task (e.g. computation)
Simulation Task (e.g. hardware IO)

Fig. 1: The **SIMUL_R PARALLEL** system.

The Language

SIMUL_R PARALLEL is the parallel computing version of the simulation language **SIMUL_R** which has been introduced in former comparisons (see comparison 1-7). **SIMUL_R PARALLEL** allows a hardware independent implementation of parallel simulation models. Submodels can be distributed to different tasks (see figure 1). The user can specify on which processor a task is placed (or the system selects it). This distribution can be optimized using **SIMUL_R**'s **DOPTCONPAR** command.

Simulation models can be distributed and started from within the Host (Runtime Interpreter), which offers the usual **SIMUL_R** desktop (optionally menu driven) with some additional commands:

start &;	start a simulation run in background.
wait;	wait till all simulation runs in background are finished.
send, receive	send and receive model and system data to/from tasks.

If one model has to be parallelized (the submodels simulate in parallel) the **SIMUL_R** translator checks which variables have to be exchanged. Nevertheless, the user is free to write easy-to-use **#SEND** and **#RECEIVE** (macro) commands to the models to explicitly exchange data values between submodels.

Task 1, Monte-Carlo study: The model for task 1 is very simple (see fig. 2)

```
parallel_1 {
  CONSTANT tend=2;
  CONSTANT m=450, k=9000, d=1000, dx0=0,
       x0=0.1;
  km=k/m;
  dm=d/m;
  DYNAMIC {
    DERIVATIVE {
      x=INTEG(dx,x0);
      dx=INTEG(-km*x-dm*dx,dx0);
    }
    TERMINATE t=tend;
  }
}
```

Fig. 2: The model for the first example

We want to use 1, 2, 4, 8, 16 tasks parallel performing 1008 simulation runs. The runtime commands - including that one for perfoming the statistics (!) - are shown in fig. 3 .

```
#set NNN=1008#
act_mod=parallel_1;          " set activated
                                    model parallel_1 "
cint=0.001;                         " set communication
                                    width "
ialg=1;                      " set RK-4 integration
                                    algorithm "
simdat_name='';              " no data file -
   sampling in memory ==> much faster "
prepare-;                           " empty sampling list "
prepare x;                          " sample x "
#loop N=1,2,4,8,16#          " use N tasks parallel
                                    at once "
  #for task=1,N #
   send dict;                " start tasks and send
                                    system infos "
  #end
  #for c=1,NNN/N #           " NNN/N * N tasks
                                    parallel "
    #for task=1,N #
     d=unif_dis(0)*400+800; " compute
                                   distributed d "
     send d;                 " send d to task "
     start &;                " simulation run in
                                   background "
    #end
    wait -1;                 " wait till all runs
                                    have finished "
    #for task=1,N #          " statistics "
     receive dict, prepare;
     #if c==1 && task==1#
       out_prep 'sum.dat';
     #else
       op2_prep '+','sum.dat','help.dat',0;
            " add the values of the new
             data file and the sum file "
       SYS 'cp help.dat sum.dat';
     #end
    #end
  #end
#end
"-> now sum.dat contains the sum of all values"
 simdat_name='sum.dat';
 op1_prep '/',NNN,'mean.dat',0;
```

Fig. 3: Runtime commands for task 1.

If *N* is the number of parallel tasks, *NNN / N* groups of tasks are started. Data is sampled in memory, which is much faster than sampling on disk. With **receive prepare** we receive these values and add them to an accumulator file *sum.dat by* **op2_prep**. **op1_prep** then divides the data values by *NNN* to get the result. As told above, this model and these commands can be started on any hardware which is supported by **SIMUL_R PARALLEL** (there is even an emulator version under MS-Windows: parallel processes communicating by Windows messages). The computations in this case are performed on the Multi-Transputer workstation Cogent™ XTM using the *Kernel Linda* system.

Number of processors	Simulation speed up	Statistics speed up
1	1.00	1.00
2	1.93	1.00
4	3.57	1.00
8	6.27	1.01
16	9.99	1.01

Fig. 4: Results for task 1

The first speed up values in fig. 4 show the simulation and model handling time relative to the 1-processor version, the second the results for accumulating data and computing the mean: here no profit can be reached.

This is a very important point, which often is not taken into account: You do not only need time for the real parallel work, but also for collecting data and displaying and storing it - and in nearly all cases displaying and storing is done on a bottle-neck single processor device (as is at the Cogent XTM)!

Task 2, coupled predator-prey population:

The coupled predator-prey population model (cint=0.01, RK4) results in a "speed-up" of 0.04. No profit can be reached using parallelization with this small model: the integration routines of **SIMUL_R** are very optimized, therefore computation goes on too fast - compared to the slow communication.

Task 3, partial differential equation:

The partial differential equation model can be modelled very easily using **SIMUL_R**'s PDE support: special macros which can translate the PDE - written down similarly to the original PDE notation $u_{tt}(x,t) = u_{xx}(x,t) / a$ - into a method for solving PDEs (e.g. the method of lines - discretization is done automatically !).

Number of processors	1	2	4	8
Speedup factor	1.00	1.79	2.75	2.35

Fig. 5: Results for task 3

The result for 8 processors is considerably bad - the Kernel Linda overhead may be the reason (generally it is not easy for the user to detect on which processor which tuple of a Linda object is placed).

Conclusions

The examples show how easy **SIMUL_R PARALLEL** can be used (hardware independently) for parameter variation tasks and parallelizing models. Some special algorithms of **SIMUL_R PARALLEL**, like the evolution strategies optimization tool **GENOPT**, can be simply used without any task start&'s and send's. **GENOPT** parallelizes itself, depending on the active tasks.

The results are not typical for **SIMUL_R PARALLEL** in general, but for the implementation on this special machine. Results for other parallel or distributed computer versions of **SIMUL_R PARALLEL** will be presented later, too.

R. Ruzicka, SIMUTECH, Hadikgasse 150, A-1140 Vienna, Austria. Tel: +43-1-894 75 08, Fax: +43-1-894 78 04.