Comparison of Parallel Simulation Techniques Multiprocessor System with Physically Distributed Memory / CPSS

CPSS (continuous parallel simulation system) is a prototype of a continuous dynamic system simulation tool on multiprocessor systems with physically distributed memory developed at the University of Magdeburg. The concept of CPSS includes the modular development of simulation models, splitting of complex models into several smaller submodels, connecting submodels, mapping them on processor nodes and simulating submodels simultaneously, with some CSSL standard features. CPSS has been developed in the programming language C based on the *extended model interconnection concept*. This concept is an extension of the model interconnection concept introduced by Schuster/ Breitenecker (mosis User's Guide, TU Vienna, 1994).

Basics: The problem when parallelizing with the model interconnection concept is the compromise between obtaining a good speed-up factor and the precision of the simulation results. The numerical inaccuracies occur if function values for the integration algorithm are needed from other submodels within the communication interval (between two communication points). These intermediate values cannot be exactly calculated. There are two ways for their calculation: using the function values from the last communication point (hold constant) or the intermediate values can be extra(inter)polated from the function values of e.g. the two last communication points (linear extra(inter)polation only with the available function values).

The goal of the extended model interconnection concept is to reduce the numerical inaccuracies when calculating the needed function values within a communication interval. Therefore the following is required:

- Three parameter values are sent at the communication point: function value *y*, the simulation time *t* at which this value is calculated and additionally the *first derivative* . The exchange of these parameters must be split into two communication instructions. First of all the function values and the simulation time had to be exchanged at the communication point. After this the right sides of the differential equations were evaluated to get the appropriate derivatives. Only now the first derivatives can be sent.
- The intermediate values can now be calculated from the available function values *y* and their first derivatives e.g. with the *Hermite interpolation algorithm*.

This concept was examined in CPSS for three explicit integration algorithms: Euler, Heun and RK4.

Hardware: Two available multiprocessor systems with physically distributed memory at University of Magdeburg were used as hardware platforms, GC/PowerPlus and the GCel 2/128 by Parsytec. GCel is a parallel system based on T805 transputers with PARIX operating system. The GC/PP also uses PARIX as the operating system (the same program development environment), but the hardware architecture is different. The GC/PP processor node architecture consists of two PowerPC-601 microprocessors and four T805 transputers. The speed-up factors for the three simulation examples explained in this article are results of the GCel 2/128.

Implementation: CPSS consists of two separate modules, the *command interpreter* and the *simulation kernel*. The interpreter (user interface) provides functions for textual inputs/outputs and graphical outputs (post run). The simulation runs can be controlled interactively by command inputs with the interpreter (e.g. setting and displaying parameter values at run time). The simulation kernel contains all necessary functions for the parallel simulation, e.g. several integration algorithms; synchronisation mechanisms for submodels (function library). The communication of the two modules is carried out via a defined TCP/IP connection (socket).

For a simulation study the user generates a specific model description file in the programming language C based on a developed parallel model structure (static partitioning and mapping). The model description file can contain several submodel definitions, each with their own specific model structure. Each submodel will be simulated on a defined processor node and can be connected to any other submodel via unidirectional links. At each communication point the parameter values are interchanged between corresponded submodels. The model description file has to be compiled and linked with the simulation kernel to an executable file. It can be run on the multiprocessor system.

Monte-Carlo study: In this example the 1000 simulation runs are distributed on available processor nodes. The simulation runs can be simulated simultaneously without communication between them, for that reason a good speed-up factor is expected. In order to solve the distribution problem the master-slave approach can be used in CPSS. The master-processor calculates the random numbers of the damping factor *d* and sends them to the slave-processors. Each slave-processor performs one simulation run with a specific damping factor *d*. If a slave-processor has finished one simulation run, it communicates with the master to receive a new damping factor *d* for the next simulation run. In Table 1 the obtained results (speed-up factors *f*) are presented for various numbers of slave-processors.

slave processor nodes	4	8	16
speed-up factor ***f***	3.98	7.92	15.6

The 1000 simulation runs were executed in the time interval [0,2] with the step width h=0.002 and using RK4.

Coupled predator-prey population: Five populations are described in this predator-prey system, which are tightly coupled. For the parallelization of this system the five populations are distributed on two processor nodes, therefore one processor node calculates three and the other node simulates two populations (h = 0.01, communication interval cint=h=0.01, RK4). Such a distributed simulation did not bring any success. The simulation on two nodes was significantly slower than in the serial case (speed-up factor *f* = 0.3). This is due

to the disadvantageous relation between calculation and communication overhead (communication calculation). A further distribution of the five populations on three or five processor nodes would produce an even worse speed-up factor, because the communication overhead is even higher.

Partial differential equation / swinging rope: Figure 1 shows the solution of the swinging rope problem for two select lines x=9L/10 and x=L/10. The simulation run is executed with a discretization of N=800 lines using eight processor nodes (time interval [0, 30], h=0.005, cint=h=0.005, RK4). Thus every processor node has to calculate 100 lines. Only the adjacent parameter values of the neighbouring processor nodes have been interchanged at the communication points (weak coupled submodels). The resulting speed-up factor was f=5.21. The non-smoothed curves in Figure 1 present the inaccurate calculation of the needed function values within the communication interval. The intermediate values were held constant. The numerical inaccuracies increase further if the 800 lines are distributed on e.g. 16 processor nodes. Figure 2 shows the solution of the two selected lines with the extended model interconnection concept. The interpolation of the needed intermediate values has been carried out with the Hermite interpolation algorithm. The resulting speed-up factor was 4.96 (8 nodes). The additional exchange of the first derivative and their use at the calculation of the intermediate values proved to be advantageous. The extended model interconnection concept shows a well-balanced relation between obtaining a good speed-up factor and the precision of the simulation results.

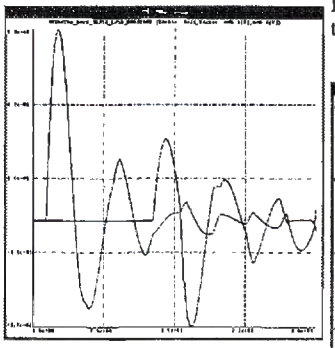

Figure 1: Swinging Rope; intermediate values were held constant

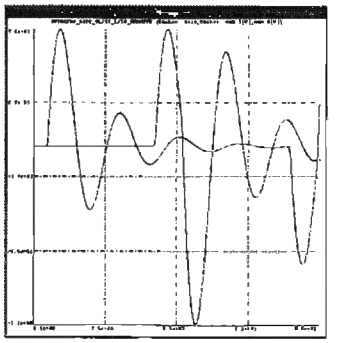

Figure 2: Swinging Rope; intermediate values were calculated with Hermite interpolation algorithm

G. Hanf, R. Hohmann, Institut für Simulation und Graphik, Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg, D-39106 Magdeburg, Germany, Email: hanf@cs.uni-magdeburg.de, hohmann@informatik.uni-magdeburg.de