Sample Solution: Workstation Cluster under PVM

All test examples of the comparison have been run on a cluster of nine IBM RS/6000-320H workstations (at the Computer Centre of the Technical University of Vienna), connected via Token Ring (16 MBit/s), using PVM version 3.1. PVM (Parallel Virtual Machine) is a software package that allows a heterogeneous network of parallel, serial, or vector computers to appear as a single distributed memory computer (virtual parallel computer). PVM consists of daemon processes and a user library. Applications (written in FORTRAN or C) can be parallelized by using message passing constructs. PVM has been developed at Oak Ridge National Laboratory and other US research labs, is distributed freely in the interest of advancement of science, and is being used in computational applications around the world.

The structure of the **Monte-Carlo study**, the first test example, permits a coarse grain parallelization in the sense of data partitioning. The basic idea is to distribute the simulation runs to the available processors. Each simulation run is independent of all others. The master processor partitions the problem, sends the initial data to and receives the results from the worker processors doing the simulation runs (fig.1). The example has been programmed in FORTRAN, single precision, using the RK4-algorithm with stepsize h=0.001.

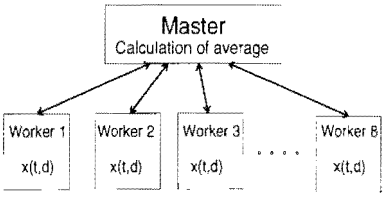

Figure 1

In order to keep communication times low as a first attempt a parallelization with static load balancing is done. Each of the eight workers in the cluster gets 125 random numbers (parameter d), performs 125 simulation runs, and returns the average response of $x(t,d)$. The master does the final computations and writes the results to a file. Postprocessing (graphical output) of the result is done using Gnuplot (see fig.2, page 21). The resulting speed-up factor with eight workers is f=5,53.

In order to gain advantages of possible free resources as a second attempt dynamic load balancing of the workers is done. As soon as one of the workers has finished one simulation run and has sent back the response $x(t,d)$ the master sends a new random number to this worker and starts the simulation again. The master calculates the average response "dynamically" while waiting for the next free worker. This procedure did not succeed. Because all 1000 responses (2000 real numbers each) are sent to the master, the overhead for communication becomes very large and the speed-up factor is significantly less than 1 (f=0.25). If the master provides for the worker more than one parameter d in advance in order to continue the simulation without waiting, the factor "improves" to f=0.95. In a second try the master controls only the dynamic load of the workers, but each worker calculates the average himself and sends it at the end to the master, which then calculates the final average response. This procedure results in a speed-up factor of f=4.9, even worse than the solution with static balancing. The worker load using dynamic balancing differs only slightly from static balancing: the workers perform 125±3 simulation runs.

Because of the homogeneous structure of the cluster and of the structure of this test example static load balancing is to prefer. Further investigations, e.g. non-homogeneous clusters with PVM or dependence of the the speed-up factor on the number of workers, will be investigated.

Parallelization of the **coupled predator-prey population** model, the second test example, was not successful, mainly because of the slow communication. As a first attempt the example is distributed in a generic way: each submodel for two populations is implemented on one worker, while the master controls only initialization of the simulation runs at the workers and has to wait for the terminal values. The workers have to exchange data corresponding to the coupling terms at communication intervals (fig.2). The example has been programmed in FORTRAN, single precision.

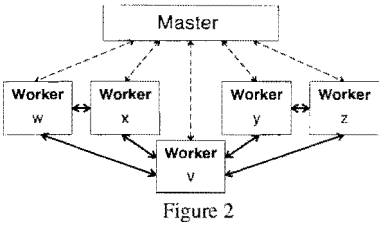

Figure 2

First, communication is performed before each integration step with steplength h=0.01 (communication interval c_{int}=h). The integration at the workers with the RK4-algorithm calculates in this case only the intermediate evaluations of the model equations with "old" values from the other workers. This parallel solution takes 20 times longer than the serial one (f=0.05). Consequently it is tried to enlarge the communication interval c_{int}, so that the integration algorithm uses "old" values for more than one step. The following table summarizes the results with communication intervals

up to $c_{int}=20h$ with sufficiently exact terminal values (deviation from solution less than 10E-4, in case of $c_{int}=20h$ 10E-3).

c_{int}	h	$2h$	$5h$	$10h$	$20h$
f	0.05	0.10	0.21	0.39	0.77

Improvements can be made by a) interpolating the data within the communication interval, b) making the communication depending on the change of the states (while some populations are still oscillating, some are almost in the equilibrium), and c) distributing the submodels to only three processors. In the last case the first worker calculates the populations v, the second w and x, and the third y and z - reducing the communication to a third (results will be published later).

The third test example, the **partial differential equation**, consists in case of a discretisation with N=800 lines $u_i(t)$ of 1600 state variables. A generic way to distribute this problem is to divide the system governing differential equations into M (number of processors) equally sized (N/M lines $u_i(t)$) subsystems implemented at one worker (fig.3). Only the boundary values (boundary lines) of each subset (worker) have to be exchanged with the neighbouring subset (worker), while the master controls only initialization and waits for the results of five lines. The problem was programmed in C, double precision, using the RK4-algorithm with fixed stepsize h=0.005.

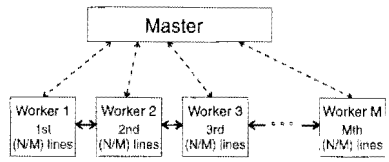

Figure 3

As in the second test example, experiments with different communication intervals c_{int} from $c_{int}=h$ up to $c_{int}=10h$ were done. Furthermore the number of lines was varied (N=600, N=800, N=1000). Using eight workers, an average speed-up factor of f=3.4 can be reached. The results for the speed-up factors are given in the following table, "*" indicates solutions with deviations greater than 10E-3, "+" instability (wrong results) because of a too long communication interval.

f	h	$2h$	$4h$	$6h$	$8h$	$10h$
N=600	0.59	1.1	1.9	2.6	3.10	3.5*
N=800	0.72	1.37	2.05	3.20	3.82*	
N=1000	0.93	2.05	3.10	3.80	4.30*	

All serial solutions were performed at the same environment with the same compilers using only one processor of the cluster. Being familiar with the PVM system, programming and testing the parallel solutions took about four times longer than programming and testing the serial solutions.

F. Breitenecker, I. Husinsky, G. Schuster