Comparison of Parallel Simulation Techniques
Cogent XTM / Linda, C

In this series of comparisons a solution with the simulation system mosis on the Cogent XTM has been described (SNE 11). mosis is a general purpose CSSL simulation language with special multiprocessing features. In this article a solution directly programmed in C using the communication system Kernel Linda on the Cogent XTM will be shown:

The Cogent XTM is a workstation consisting of up to 32 transputers T800 under a distributed UNIX-like operating system called QIX, featuring parallel processing (even the operating system kernel is distributed over the network) and PIX, a PostScript input/output system with possibility to run X Window applications. As a communication standard Kernel Linda being a derivate of Linda is used, even for system process communication. Each transputer works with 20 MHz and is equipped with 4 MB of local memory. The main console (the minimum configuration) of a XTM workstation consists of two transputers that are linked to the graphical I/O system and that control the hard disk access. This system can be expanded to a maximum number of 32 by connecting to a "resource server" containing up to 15 boards with 2 transputers each. For inter-processor communication two different media can be used: Usually messages are sent via the transputer-unique so-called "Links" (serial, 20 MBit/s transfer rate). For short messages a very fast bus system is installed.

Description of Linda: Linda is a programming model for parallel algorithms that was initially developed by David Gelernter at Yale University. In this model, all processes communicate with each other by accessing a common "tuple space" which is logically like reading from a shared memory area, but physically implemented by message passing. A message can be sent to another process by putting a "tuple" into the tuple space (function "out") that can be read by any other process that has access to this tuple space ("in" reads the tuple and deletes it; "rd" reads the tuple without destroying it). The tuple contains an identifier (usually a name or an integer number) and a data area.

Within the tuple space several tuples with the same identifier may be defined. In this case, at the "rd" or "in" operation that was written first, is read. All tuples with the same identifier are located in a FIFO queue, which can be also used for message passing.

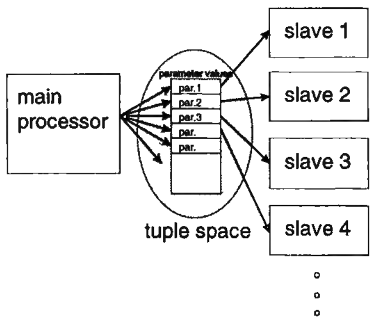
Kernel Linda is an implementation of this programming paradigm by Cogent Research for the Cogent XTM where the source files of parallel programs need not be passed through a preprocessor.

Solution of the Comparison Examples

All tasks were solved by using the Runge-Kutta fourth order algorithm, with fixed stepsizes, depending on the particular problem. All floating-point variables were defined as "double". For the solutions the "shared memory" approach combined with the usage of FIFO queues for tuples with the same identifier was used.

Each task was programmed in a serial fashion which was executed on one single processor and in a parallel version using several concurrent processors. In examples 1 and 3, only eight slave processors were used for comparability to the other solution given in this series (workstation cluster connected with PVM, SNE 10).

The solution for the **Monte Carlo Study** uses a master-slave approach with *dynamic load balancing* (in contrary to the PVM and mosis solutions that use static load balancing) as this can be programmed in Linda in a very elegant way. This means that when one processor has finished one simulation run, it can immediately start with the next one (using a different parameter value). At the main processor, after creation of several (eight) similar processes on different processors, the desired number of random numbers is created and written into a FIFO queue in the current tuple space. Each processor reads (and deletes) one tuple and performs the simulation run. As soon as this has completed, the next tuple is fetched until no value can be found in the queue. Then the sum of all simulation runs within this processor is evaluated and sent to the main task (by putting into the tuple space) which calculates the average of all runs. The following figure illustrates the distribution of tasks:



As a calculation base, 1000 simulation runs were performed by 8 slave processors. The achieved speed-up factor was *f*=7.8. For this homogenous system, static load balancing would probably produce similar results.

The distributed simulation of the **Coupled Predator-Prey model** resulted in a "speed-up" factor of less than one, i.e. the parallel version was significantly slower than the serial one. Communication is done via "global variables" (tuples in the environment) containing the current state of the coupled model. The five tasks representing the various populations were simulated on different processors.

The resulting "speed-up" factor was f=0.08 which could be improved to f=0.60 by communicating only each 10th integration interval h (c_{int}=10.h). This problem seems to be not yet suitable for parallel processing with distributed memory, shared memory structures may give better results.

The third test example, the parallelization of the **partial differential equation** (PDE) proved again to succeed in terms of calculation speed. The model was calculated using N=600 and N=800 discretisation lines for the PDE; the latter produced even better speed-up factors. The model was simulated in the same way as in the solution on a workstation cluster under PVM (SNE 10, p.24) with eight concurrent processors (each calculating 75 or 100 lines of the PDE) and produced speed up factors summarized in the following table.

# Lines / communication	$c_{int} = h$	$c_{int} = 4\,h$
N = 600	f = 6,78	f = 7.54
N = 800	f = 6,88	f = 7.62

Communicating only each fourth integrating step improved the results up to an almost linear speed-up.

Summary of the Results: The solutions directly programmed in C and Kernel Linda produce slightly better results than those programmed in C and PVM on an RS6000-cluster and those on the Cogent XTM using mosis; on one side this is because the ratio of communication by calculation speed is higher than on the workstation cluster (faster communication, slower calculation), on the other side the C-programs do not have to poll for incoming messages (which makes the simulation with mosis slower, but which will be improved).

But the advantage in simulation speed must be paid by much higher development cost: A mosis model needs only be compiled and run on the parallel computer system, but the implementation of this model took quite a long time (approx. one week for implementation, testing and simulation), although the existing PVM models could be used and had only to be transformed to Linda programs.

G. Schuster, F. Breitenecker, ARGE Simulation News, c/o Dept. Simulation Techniques, TU Vienna, Wiedner Hauptstr. 8-10, A-1040 Vienna, Austria, Email: argesim@simserv.tuwien.ac.at.