Experiences in PVM-based power plant simulation

A. Femminella, A. Omodeo
Unità Tecnologie dell'Informazione - Centro Ricerca di Automatica, ENEL
Via Volta 1, I-20093 Cologno Monzese (Milano), ITALY
e-mail: {femminella|omodeo}@cra.enel.it

ABSTRACT

In this paper, we report our experiences in designing a parallel version of a power plant simulator on a workstation cluster.

Due to the absence of standards in this field, our activity was focused upon the implementation of a distributed software platform, open to heterogeneous systems, and to be used as a reference for our future developments.

After a brief description of the simulation environment, we emphasize the limitations emerged in a first parallel version based on a Transputer network that led us to the current evolution based on a network of heterogeneous workstations.

Finally, performance results are reported in synthesis, and future plans are outlined.

1. DYNAMIC SYSTEMS SIMULATION

Simulators are an effective aid in the scientific and industrial world, where they are used in many design activities and in the training of qualified personnel.

A training simulator allows one to know the response to a particular manoeuvre, to evaluate its behaviour and take, when necessary, the needed actions to re-establish safe conditions.

An extensive effort has been made at ENEL (the Italian Electrical Power Company) to develop LEGO [1], a system for computer aided modelling of fossil-fired and nuclear steam power plants.

The basic characteristic of LEGO are:

- **modularity**: base modules for regular plant components (such as pipes, valves, pumps...) are available in a library of modules and can be linked by the user to simulate a specific plant;
- **flexibility**: the user can add ad hoc models to solve specific problems and connect them to existing modules;
- **reliability**: all the numerical algorithms are centralised in a master database and the development of new modules does not require any algorithm updating;
- **availability** of the models to users without specific modeling knowledge.

The LEGO system has been extensively used in the design of the power plants of Brindisi Sud and Tavazzano.

The user interface is a set of animated diagrams, each one representing a plant subsystem, and some software panels emulating the control panels of a real power plant.

A user can interact with the system by touching, with the mouse, the virtualized components of the panels (figure 1).

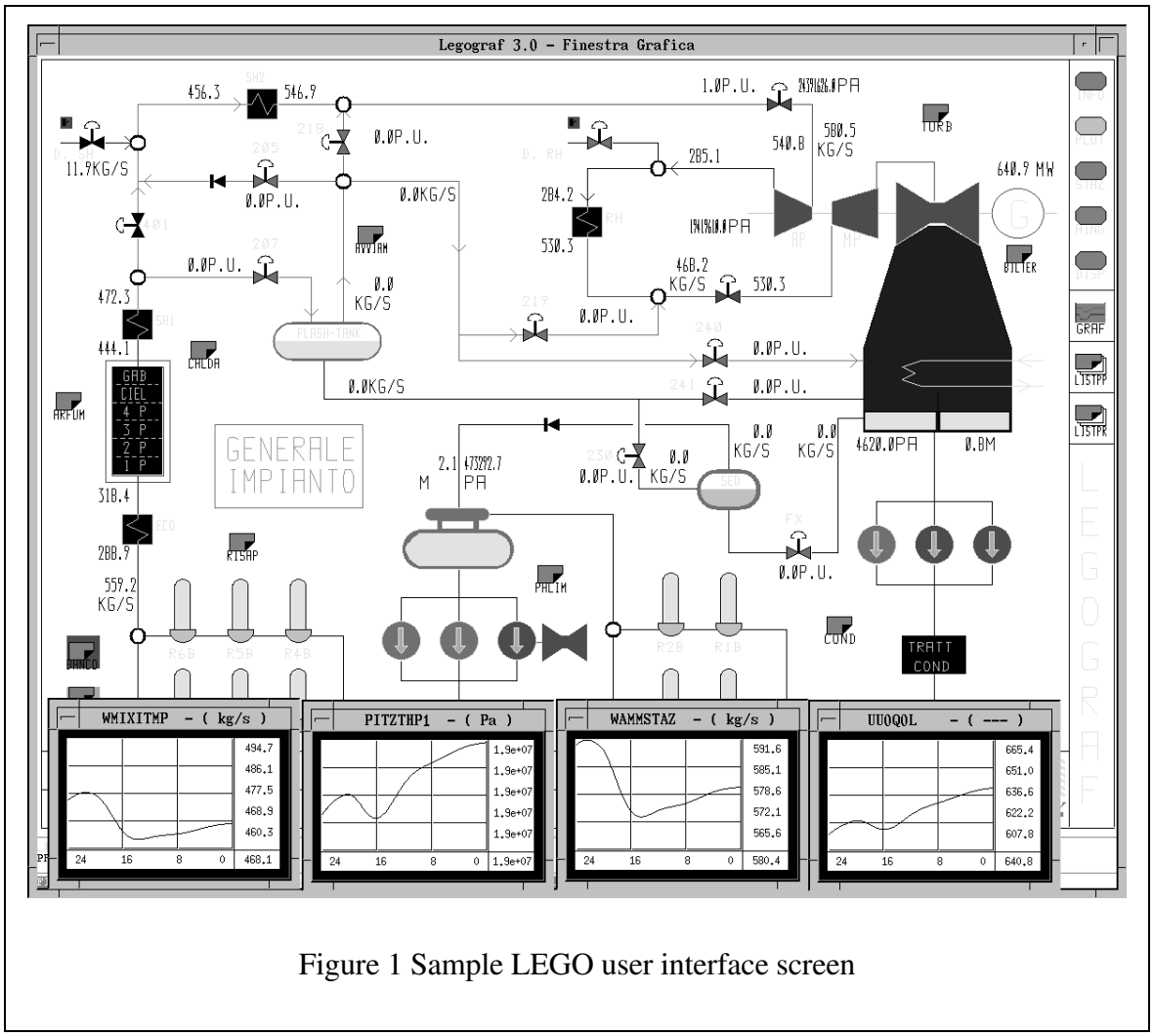

Figure 1 Sample LEGO user interface screen

The power plant simulators realized with LEGO are built by connecting the process variables of each module with the ones of other modules, where each module simulates a physical plant component.

Each plant subsystem is a separate UNIX task: another process (called scheduler) synchronizes all the processes according to a real time clock and exchanges the input/output variables of one process with another.

The scheduler has specific commands for simulation control, like *initialization*, with the option to restart a previous simulation (*snapshot*), *run* and *freeze* to start and to halt the simulation, *step* for manually advancing, time scaling control to speed up or slow down the simulation with respect to real time as well as the recording of selected variables.

The base version of the program runs on a single machine where the simulation data is stored by the processes in a shared memory area. The scheduler updates the input and output variables of submodels, synchronizes the slave tasks according to the dynamics and restarts a new cycle.

Given the strong uncoupling between the tasks, we planned to implement this on distributed memory systems, first in a dedicated environment like a Transputer network and then in a workstation cluster using PVM.

2. THE FIRST PARALLEL IMPLEMENTATION: LESSON LEARNED

To test the simulator we chose the model of a real fossil-fueled power plant, a 660 MW station in Brindisi Sud [2].

The system organisation involves:

- 11 simulation processes, one for each power plant submodel;
- 3 auxiliary processes for control and synchronisation of submodels;
- up to 5 auxiliary processes for the man-machine interface.

The hardware platform is composed of 16 T805 microprocessors, 32 bit RISC, with a floating point unit and 4 MB of local memory, linked to a SUN4 workstation for the I/O services (X terminals, LAN Ethernet, hard disk..).

The porting of this LEGO simulator was an excellent practical result but was limited from the strategic point of view; in fact the simulator is bound to the Helios operating system and it doesn't allow the porting to newer architectures. Moreover, the new generation of Transputer didn't evolve as we expected when we started with the project.

To avoid this situation in the future and to ease the building and porting of parallel computing projects we had the problem of choosing a parallel programming standard to communicate among programs allocated on a cluster of computers. We decided to use PVM (Parallel Virtual Machine) for our work. PVM is a public domain tool developed by the Oak Ridge National Laboratory (Tennessee) and it is becoming a standard for programming distributed environments [3].

3. A PVM - BASED IMPLEMENTATION

PVM is a library for programming many distributed memory systems and workstation clusters connected by standard network software (TCP/IP).

Starting from the first release, PVM was written in the C language, to allow greater portability on every system running the UNIX operating system. Programs using PVM can be ported to a wide variety of systems, from microprocessors to supercomputers, allowing the user to choose between low performance, and low-cost hardware, or high-performance and higher cost machines [4].

PVM is currently available on a wide choice of computers, and this is a key advantage because our simulator, apart from CPU-bound tasks, needs some auxiliary user interface processes.

Given this situation, it is believed that the best solution could be a mixed-architecture environment, with some processes running on bare-bone processors with a minimal run-time support, while man-machine interface processes are running on graphic workstations with suitable hardware.

The number and type of processes running in the PVM version are the same as in the first version we implemented. In the base version the computing processes share a common memory area used for exchanging program variables. At every simulated time step each process writes its results in this area as a floating point vector and reads the input variables for the next step. The shared memory accesses are very regular so it was easy to adapt this communication behaviour to a message-passing programming environment.

Other tasks, principally the user interface tasks, exchange data via Interprocess Communications (IPC), a Unix facility to implement message passing between processes

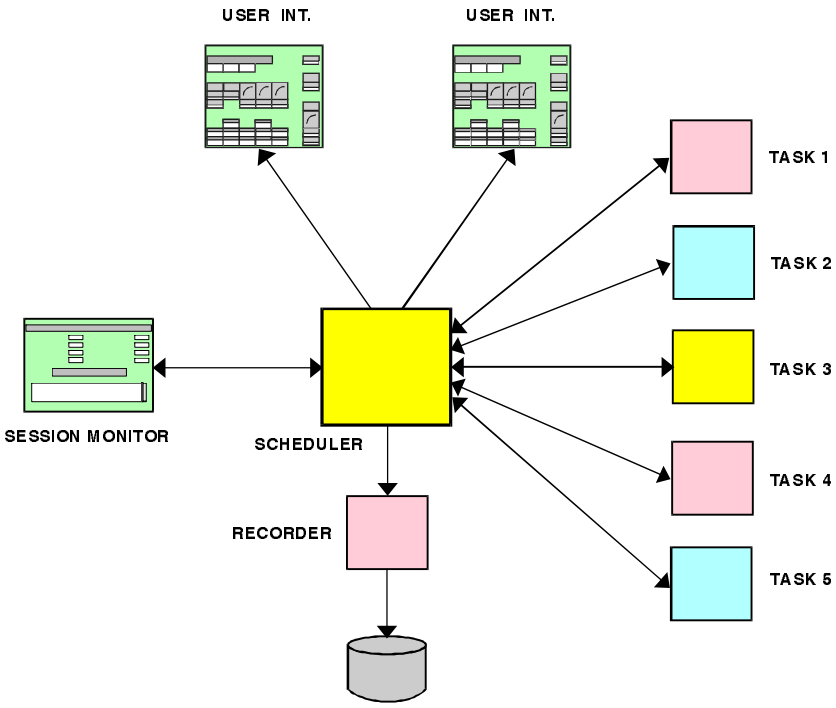

Figure 2 Process structure of a LEGO simulator

running on the same host (figure 2). After converting in PVM these calls it is possible to activate user interface tasks separately on different machines.

There are significant differences among the 14 tasks which constitute our simulator. *Simulation tasks* are written in Fortran and perform a lot of floating-point computations while *user interface tasks* are written in C and are mainly made up of integer operations.

Again, simulation tasks can be divided in *plant systems* and *control systems*.

A control system is a relatively small task which performs a deterministic number of floating-point operations, while a plant system task solves a non-linear equation system using an iterative method. Because of this iterative procedure, it is impossible to predict how much time it is necessary to complete a single simulation step. In general, simulation tasks require more processing time when the simulator is computing abrupt state changes.

Another difference among the tasks is the number of variables exchanged with the master process (called *scheduler*). The control system processes exchange a great percentage of the state variables, while plant systems generally have a much smaller impact on communication load.

4. EXPERIMENTAL RESULTS ON AN ETHERNET NETWORK

At present, the hardware architecture of our laboratory includes two DIGITAL AXP 3000/600 workstations, an IBM RS/6000-370 with a graphics accelerator and an IBM RS/6000-340.

These machines are different both in the hardware and in the system software, and it is not possible to use simple benchmarks (like Dhrystone or even Linpack) to define how much a machine is faster than another for our computing needs.

To rank the performance of our machines while running LEGO code, we mapped all the processes on a single machine, and then measured the fastest achievable simulation speed[1]. Table 1 shows the maximum speed obtained by the simulator running on a single workstation.

Machine	Speed vs real time
IBM RS/6000 - 340	1.14
IBM RS/6000 - 370	2.69
DIGITAL 3000/600	3.50

Table 1: Relative performance of the machines

Figure 3 summarizes the relative CPU (grey bars) and communication loads (white bars) of every process (it is not possible to exactly define the communication load for user interface processes, but some measures shows it to be negligible).

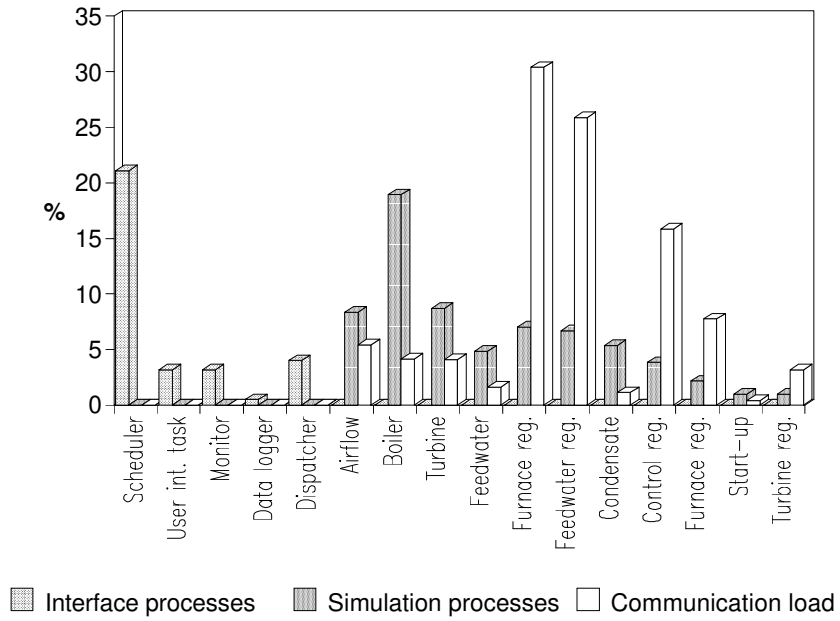

Figure 3 CPU and communications loads for LEGO processes.

From the figure, we can easily distinguish between CPU-intensive tasks (where the grey bar is higher than the white one) and communication-intensive ones.

[1] Every test case referred in this article starts from a recorded state, where the plant is generating 630 MW. From here, we cut power production down to 430 MW with a ramp of -10 MW per minute. We define as speed factor the ratio between the "simulated" time and the real time.

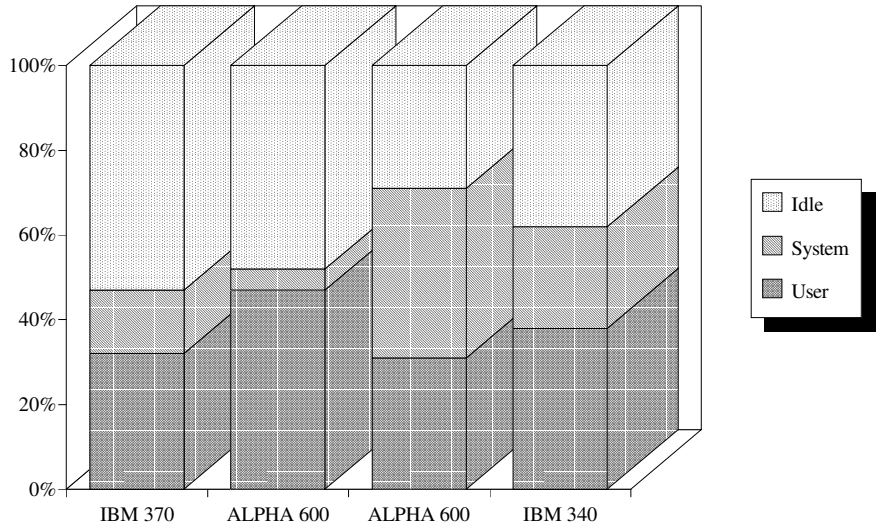

Figure 4: CPU utilization during simulator activity.

Since we have a set of 14 heterogeneous processes to be allocated on 4 heterogeneous computers, the problem of task allocation is not so trivial and has a great impact on the performances of the simulator.

Since PVM doesn't have an automatic tool to balance the load on the computing nodes, we should assign the tasks on the available machines basing our decisions on the experimental results of Figure 3.

The key process is the scheduler, and it should be allocated on the fastest available machine. On the same machine it is preferable to execute the control systems with small computing load but performing great communication exchanges with the scheduler. Other processes can be spread out on the remaining machines, preferably keeping user interface tasks on the machine with the graphics accelerator.

The PVM library allows the programmer to select between direct (TCP-based) and indirect communication, requiring the PVM daemons in the interested machined to perform the data exchange via UDP sockets. This option can be set on a per-process basis and can have a big impact on system performance.

After trying the possible combinations, it seems that direct communication is preferable for simulation tasks, because the performance of these communications is essential for the simulation to run. By contrast, it is better to choose routed (indirect) communication for the user-interface tasks, because routed communication frees the scheduler to continue its execution before the message is arrives at the receiving task. After all, we can accept a small delay in the visualization of data if this enables the scheduler to continue with execution.

In Figure 4 we can see the relative CPU activity imposed by the distributed version of the simulator during the transient simulation formerly described.

CPU loads of figure 4 are obtained by the **vmstat** Unix utility and have the following meaning:

- *user time* is spent by the CPU while processing user code - in this case our application and the PVM daemons;
- *system time* is spent while processing operating system code, mainly the socket-level communications imposed by PVM;
- during *idle time* the processes of a specific CPU are standing by, waiting for input from processes running on other machines; this CPU time is available to other users.

Idle time is the limiting factor for simulator speed, but it is not a completely negative factor. First of all, idle time is a spare computing reserve so that the simulation speed is not slowed down when we require heavy transient computations.

The high idle time on the graphical machine allows us to enhance the user interface programs without disturbing simulation performance.

As a summarising result, the maximum simulation speed obtained is 5.89 times faster than the real time, both on steady and on transient conditions. While this speed is not impressive, it is a good improvement compared with the speeds achieved on single machine simulations.

If we look at the high idle time on the four machines, we should consider: a) the presence of serial bottlenecks in our code; b) the limits of the physical network connection. Primarily we investigated network problems, thinking they were the limiting factor for application performance.

In our case, using an Ethernet network analyzer, we see communication rates under 20% of the available bandwidth, so Ethernet bandwidth is not a constraint for our application. Since Ethernet is a contention protocol and our application relies on the synchronization of slave processes, we should also consider the case where all the slaves perform short network accesses at the same time, causing high network collision rates. Again, on our network, collision rates are under the measurable threshold of 0.05% for very long periods of time.

The serial bottleneck in our application is in the scheduler code: the scheduler has to wait for data from all the processes, then it distributes input data to the slaves, restarting a new cycle. After the slaves are restarted, the scheduler performs less urgent operations like sending updated data to the user interface tasks.

In the single-machine implementation the computing load of the scheduler was very low, but this is not the case: in the parallel implementation the scheduler is requested to unpack, store, pack and send data to every slave. From our single-machine measures (see figure 3) we see that the scheduler is now the most computing-intensive process.

Even when considering the serial bottleneck of the scheduler, this is not enough to justify the measured idle times.

Of course, we can consider the scheduler as a complete serial bottleneck only in a worst-case analysis: the high idle times we see are probably due to the scheduler, to the latency introduced by the network media and to interactions with operating system tasks running on the machines.

We are going to investigate in further detail this problem moving this application to the SP/1 machine which provides a homogeneous parallel environment.

5. EFFECT OF A FDDI UPGRADE

After concluding the measures on the Ethernet configuration, the network connection of our laboratory has been upgraded to FDDI, including three of our test machines (but excluding the

IBM 370 which is still connected via Ethernet). Where possible, the machine configurations were tuned for optimizing TCP/IP performance over FDDI.

Even if the network media is 10 times faster than Ethernet, the PVM benchmarks on this configuration were not promising, giving the same communication latency of Ethernet and only a 20% decrease in communication times for small (under 4 Kbytes) size packets. Better results are obtained for messages of greater size, but in our application the few processes exchanging messages of this size are already allocated on the same machine.

The simulation performance is almost identical to the one obtained with Ethernet, with some small variations on CPU loads. We can only see that a non-optimal assignment on tasks now gives smaller performance degradation.

6. FUTURE WORK

In another ENEL Research Center there is installed an IBM SP/1 multiprocessing system with 8 nodes connected by a proprietary high speed network. An improved PVM version customised by IBM runs on this machine. We are porting the simulator on this architecture and the results will be available in short time.

When the SP/1 results are available, we can consider a new implementation of the scheduler to lower synchronization times.

ACKNOWLEDGEMENTS

Thanks are due to G. Garbossa who made available the current version of LEGO and co-operated to the Helios implementation and to G. Villa, the LEGO expert who helped us setting up the test cases for LEGO simulations

Finally, we would like to thank D. Lucarella who launched, supported and advised this research project.

REFERENCES

[1] G. Garbossa, S. Spelta: A model of Alto Lazio boiling water reactor using the LEGO code balance of plant simulation *7th Power Plant Dynamics, Control & Testing Symposium*, Knoxville, Tennessee, May 1989.

[2] A. Femminella G. Garbossa: Simulazione di impianti termici su Transputer. *Automazione e Strumentazione*, November 1993 (in Italian).

[3] A. Geist, A. Beguelin, J. J. Dongarra, W. Jang, R. Mancheck and V. S: Sunderam. PVM 3 user's guide and reference manual. *Technical Report ORNL/TM-12187,* Oak Ridge National Laboratory, May 1993.

[4] A. Geist, V. S. Sunderam: Network Based Concurrent Computing on the PVM System *Concurrency: Practice and Experience*, 4(4) pp. 293 - 311, June 1992

[5] PVM-based parallel computing: a case study on power plant simulation. *Short Notes Proceedings of the 1994 EUROMICRO Conference* Liverpool, September 1994 (in press).

Offending Command = .
Error = nametype : undefined
this name is not defined in a dictionary
Stack =