Transactions on Information and Communications Technologies vol 9, © 1995 WIT Press, www.witpress.com, ISSN 1743-3517

Using workstation networks for parallel computing

A. Genco[a,b]
[a]*Dipartimento di Ingegneria Elettrica, Università di Palermo, Viale delle Scienze, 90128 Palermo, Italy*
[b]*Centro Studi sulle Reti di Elaboratori, Consiglio Nazionale delle Ricerche, Viale delle Scienze, 90128 Palermo, Italy*

Abstract

The design and the execution of parallel applications on heterogeneous workstation networks are discussed. Two basic topics of the paper are: the load balancing matter, and the data exchange overhead into a loosely coupled system, in particular, when communication is performed by LAN media.
Some solutions are proposed that can be implemented by application code, as well as by system code or both. Finally, the advantage of using workstation networks is discussed in relation to the different types of algorithm and related parallel implementation schemes.

1 Introduction

The early approaches to parallel computing were carried out on machines whose cost were very high. This was the case of the famous Illiac IV whose only exemplar was sold for forty million dollars in 1972 . Not very different the case of the early Cray or Cyber supercomputers whose cost was higher than then million dollars. And this is still the case of vector or massively-parallel systems of our times. Partly due to this reason, some researchers carried out many studies that were leading to purely-theoretical results; the increase of the degree of parallelism was the only goal to achieve and no implementation problem was considered, with no care to the fact that a too fine-grain parallelism entails a lot of communication between processors.

In recent years, the financial reasons of researchers have pushed computer factories to produce low-cost parallel systems that are sold even in the shape of PC-boards with a certain number of processors mounted. Transputers, for instance, have been, and can still be considered, a powerful parallel option instead of the sequential-main processor of PC (early T800 was delivered nearly

in the period of the I286). Other low-cost microprocessors (e.g. Zoran, I860) have been designed for vector-processing in a PC board. PC power has been increased notably, and some parallel features have been included in the design of the last-generation microprocessors. This is the case of the Pentium processor with two computation units. These growing potentialities often worry transputer users; they spent a lot of time in developing parallel applications for those systems, and now have to decide if it is more convenient to buy more performing (and expensive) transputers, such as T9000, or completely give up the parallel approach.

The continuous technological improvements in conventional computer systems renew this old question to all parallel application programmers, and the most usual answer keeps on saying: whatever be the technology of sequential processors, multiple sequential processors perform better than only one, if they can exchange data. This is a good answer to the general question, but it does not give any suggestion on how to implement parallel computing. Are parallel systems really necessary in all the cases ? In other words, do our parallel applications really require a so called supercomputer for execution ? In most cases the minimization of execution-time is only intended to achieve a better throughput, so that productivity can be improved. This is the most common need, both in research laboratories and in professional offices. Just a few institutions, or private centres, really have to tackle problems for what time bounds are actual constraints and for which the use of supercomputers is a must.

Another important problem for the early parallel applications was the communication medium. It has a double influence on parallel performance: One consists in that communication statements often acts as synchronisation barriers. This means that the co-operating processes should arrive at the communication rendezvous at the same time; otherwise the waiting period of the faster process inevitably affects the efficiency of the parallel application. The other influence is determined by the cost-performance features of the parallel systems. Strictly-coupled processors were very expensive and the loosely-coupled ones of that period performed too slow communications. If we consider that most researchers, just like nowadays, worked with a few funds, the data exchange problem was an actual one.

2 Workstation-network limits

When we plan to develop parallel applications, an early option to consider is whether to buy a parallel system or to become an user of a parallel-computing centre, thus avoiding installation problems. If we deal with very expensive systems such as Cray, Connection Machine, Ncube, Masspar, Paragon, and so on, this aspect is not a true problem in most cases. Their cost is so high that only a few organisations can consider to buy one. In some cases, as for the hyper-cube architecture, it could be affordable to buy a low order hypercube (for instance, four nodes only) even in the shape of PC boards. However, their cost is often higher than transputer boards indeed, and therefore, these

configurations are mainly intended as development systems for application to be executed on bigger cubes.

The cheapest parallel system that we can consider is the one we can arrange connecting the workstations available in our configuration. We can do this by means of a LAN However we cannot consider the workstation network the exact equivalent of a parallel system. There are some problems that arise with evidence.

One problem is in the data transmission rate. Then megabits per second, (the transmission rate of an Ethernet LAN), are not a low performance in absolute; for instance, they are half the rate of the T800 links. However, we should consider that this is just a nominal peek-performance rate. LAN bus is a shared medium and all communications perform sequentially. The actual bandwidth can be estimated as an average quantity never allowing us to exactly predict how long a communication may take.

The other problem deals with the parallel application design. The early approaches to parallel computing were characterised by the partitioning problem. Many sequential applications were reviewed to detect an efficient domain decomposition so that different parts of data or code could be processed by different processors at the same time. Load distribution was a very relevant factor in the performance of parallel applications, because parallel systems were built connecting homogeneous processors. With these systems, in particular when a processor just works for one process at a time, if one of the co-operating processors is loaded with a longer part of the work, the whole application turns out to be delayed. Therefore, transforming a sequential application in a parallel one, mostly means to detect a suitable division of the application domain by equal parts, so as to achieve an efficient load balance of the work among the available processors.

In the case of workstation networks, it is very rare that the workstations we bought during the last years, all are of the same type and power. The approach to a distributed system, that is made up of heterogeneous machines, cannot be the same as in the case of true-parallel systems. It is quite an ineffective solution to distribute the working load of an application in equal tasks, with the same computational complexity, and put each of them in different machines; the time performance of the application will be inevitably affected by the slowest machine. Furthermore, even in the case of equivalent workstations, we should consider that each of them has a different owner or user, and therefore, the distributed applications have to share the resources with the owner's processes that are locally scheduled. This means that now the unbalancing factor also comes from machines as well as from parallel application design.

Under these hypotheses, load distribution must take into account the different processing capabilities of each used workstation. Some benchmarks should be carried out to detect a performance index for each of them. However, the different hardware features, and in particular the memory capacity, have a relevant influence on the scalability of the application. When we increase the number of processes, we also increase the paging activity, and in some cases, the operating system may fall in a trashing state. Therefore when we are executing benchmarks, we should consider that we may obtain different ratios

between the performance indexes, depending on the size of the application to be run.

A further aspect to be considered is the different data-representation formats as implemented by the different computer factories. For instance, DEC data formats for the Alpha workstations are different from those ones by IBM and SUN [1]. These differences entail a conversion overhead for each communication to be performed between machines with different data formats. This overhead must be considered when distributing the working load among the machines.

3 Distributed application design and implementation

Due to the above unbalancing factors, the main question to be solved in the design of a distributed application is how to distribute the co-operating processes. To obtain an efficient solution, we should follow a suitable schedule as the one listed below. It summarize the above considerations giving a sequence of the related steps to carry out.

1) Determine the size of the problem;
2) Run the application sequentially in your workstation (the one that you will use as the I/O server), and record the turnaround time;
3) Design the parallel application distributing the working load among a suitable number of co-operating processes;
4) Select the workstations to be used for execution, avoiding too unbalanced configurations. Including too low performing machines into the distributed system may entail performance downgrading in many cases.
5) Run the parallel application sequentially (with all the processes on the same machine) in each machine, and record the turnaround times;
If the nature of the application allows you to reduce the number of processes, carry out step 5 reducing the number of processees by a factor equal to the number of used machines;
6) Use the obtained turnaround times to determine a performance index for each machine;
7) Execute the application, distributing the processes among the used machines according to their performance index;
8) Carry out some more trials to detect the best distribution of processes among the used machines;
9) Determine the relative speedup comparing the sequential and parallel execution times.
10) Determine the efficiency of the parallel execution dividing the speedup by a number that represents the increase of processing rate that has been totalized by the distributed system. It can be calculated as the sum of the performance indexes of the used machines, divided by the performance index of your workstation, were you ran the application sequentially.

Transactions on Information and Communications Technologies vol 9, © 1995 WIT Press, www.witpress.com, ISSN 1743-3517

All the above steps should be repeated each time we change the application or its size, because, as said above, the performance indexes can vary depending on the memory capacity. In the case of very large applications, step 2 may have problems due to memory capacity limits, that will stop our workstation from hosting the whole application. In these cases it will be very difficult to evaluate the performance improvements, but the possibility of executing the application in the distributed system will surely represent a further advantage in addition to time performance enhancement.

When carrying out steps 9 and 10, comparisons can be performed if both sequential and parallel solutions give the same numerical result. In the case of optimization problems, or in general, when we track approximated solutions, we should include some statistical measures in performance evaluation, so as to take into account the possible improvements in the results that are achieved by the parallel solution. This last consideration can be extended to all parallel systems, but it is particularly relevant for workstation networks. Heterogeneous distributed systems are more effective when executing asynchronous algorithms as optimization problems often requires.

Some other strategies can be adopted when implementing the distributed solution. For example, we can consider whether to create a process for each partition of the application and adopt a suitable distribution of them among the machines, or to create a process for each involved machine, making this process carry out the equivalent amount of working load. The former solution has the advantage to take a bigger share of the cpu time on the same machine. This happens in Unix systems were the short-term scheduler assigns the time slices equally, so that, supposed n the number of processes of the other applications already running in the workstation, if we create m processes, our time share can be evaluated as $\frac{m}{n+m}$ that is greater than $\frac{1}{n+1}$ as in the case of only one process created by our application.

Opposite to this consideration is the fact that a large number of processes entails more scheduling overhead, more communications through the LAN bus, and in general, a larger amount of memory required.

4 Operating system facilities and user requirements

According to these considerations, efficient solutions for distributing a parallel application into a distributed system seems to be very complicate. Furthermore, if we want to exploit heterogeneous distributed systems, once detected the most suitable process distribution among the connected machines, we must also consider that a static allocation of processes can be the best one at the begininng of the execution, but may become ineffective later, if the cpu contention of some machines changes, because of other incoming applications. For this evenience, a better performance can be achieved putting a mechanism in the application code to migrate processes and redistribute the working load. Each machine will be shared among many processes of the same application or

different applications, with the possibility of making processes migrate, in order to guarantee the performance requirements of the applications.

This furterly complicates the development of distributed applications, and therefore, we would be very happy if the operating system was capable of solving the problem. There are three possible options to consider: the migration mechanism can be dynamically performed either by application code, or by system code, or both. Actually, the operating system can do this, but only if it is suitably driven by application directions.

Scheduling activity must take into account the performance indexes of the machines in order to suitably allocate tasks among them. However, besides the different computation capabilities and the dynamic variation of the working load, the allocation strategy has to consider the synchronization constraints of the co-operating processes. In other words, the true problem does not necessarily deal with the perfect partitioning of the application, but with the delays that may occurr at each synchronization barrier. A process of whatever computational complexity may perform fast or slow in dependence of the fact that it is running on a higher or lower performing machine. However, a process can be surely considered as a fast one if it is forced to wait for a communication with another process. Such an occurrence can be detected by the application code but not by the system scheduler. Therefore, a co-operation between application and operating system will be needed to detect delays, and carry out the needed migrations. This will allow each process to run in the appropriate machine.

To do this, the problem is moved to the distributed system design, involving long, medium and short therm scheduling as summarized in Table I. The local scheduling scheme mostly refers to Unix systems, while some global scheduling features can be found in distributed systems such as Amoeba [2], Condor [4], Express [3], Linda [5], and PVM [6].

Similarly to sequential operating systems, long term distributed scheduling must adopt some priority criteria to mediate user requirements. A user objective may conflict with the global criteria of resource optimization; in particular, the scheduler could make decisions that, at the same time, enhance the performance of a single application and downgrade the other ones. Preferring the requirements of an application is a matter of priority that should have a counterpart in the cost that the user is willing to pay for. A user may be interested, for example, in the overall execution time (turnaround time), either keeping it as low as possible or specifying an upper limit (deadline). Another concern may be having the application executed with a specified reliability that can be achieved by means of process replication.

Table I: Distributed scheduling structure.

Scheduling	Local	Global
Long Term	Initiator	Job acceptor • resourse check • module replication • initial allocation
Medium Term	Swapper	Distributed mimgration manager • process relocation Distributed migration carrier • process transfer Distributed information manager • status update • status brodcast
Short Term	CPU scheduler Process dispatcher	Application monitors • performance evaluation Stand-alone application tutor

5 Algorithm categories

It should be considered that the earliest parallel applications mainly were classical numerical algorithms such as finite element methods, partial difference equations, matrix operations and so on, all of them being deterministic algorithms, all of them being originally conceived for a sequential implementation. For this class of applications the parallel design almost always adhered to one of the two most used types of parallelism, space parallelism and time parallelism, that can be implemented by means of suitable partitionings of data or code. There is another type of parallelism that we can call target-shooting. This includes all those algorithms whose result is an approximation of the optimal solution. Many operation research methods fall in this category, in particular when the search is driven by heuristic criteria. For instance we can consider the Beam Search algorithm, or the most of natural methods such as Simulated Annealing [7], Tabu Search [9], Genetic Algorithms [10], and so on. Each their execution is a trial that can be completely independent each from the other or can implement some co-operation strategies. In any cases we deal with parallel processes that mostly execute in an asynchronous or loosely synchronous way. This makes heterogeneous distributed systems really suitable for parallel applications, because the synchronisation problem becomes a minor one. We can load the different workstations with a different number of processes with a few communications that mostly perform asynchronously [8].

6 Conclusions

The main advantages and problems of using a workstation network for parallel computing have been discussed. The most relevant considerations highlight the

differences between conventional parallel systems and distributed systems, in particular when multiple applications have to share the same machines.

Some strategies for designing and implementing distributed applications, with suggestions to achieve a suitable distribution of processes, have been proposed. To this end, the co-operation between application and system scheduler has been considered. This allows processes to migrate towards the most suitable machines. The proposed mechanism takes into account the delays that a high performing process may undergo when communicating synchronously with a slower process. Workstation networks turn out to be very effective when executing asynchronous or losely synchronous distributed algorithms. A further advantage can be found in simulating parallel applications. Nevertheless, the most interesting aspect is that parallel computing becomes very popular, since it is very cheap to implement, for instance, by means of some workstations or PC's, an Ethernet LAN, and public domain packages as Linux and PVM.

Acknowledgements

This study has been partially supported by MURST, the Italian Ministry of University and Researches for Science and Techology and partially by CNR, the Italian National Council of Researches.

References

1. SunSoft, SunOS 5.3 Network Interfaces Programmer's Guide, XDR Protocol Specification, pp 267-288, part number 801-5295-10, november 1993
2. Tanenbaum *et al.*, Experiences with the Amoeba Distributed System, *Commun. of the ACM*, vol. 33, pp. 44-63, 1990
3. ParaSoft Corporation, Express Fortran User's Guide, version 3.0, 1990
4. Litzkow, M. Livny, Experience with the Condor Distributed Batch System, *Proc. of the IEEE Workshop on Experimental Distributed Systems*, Huntsville, AL, 1990
5. Carriero, D. Gelernter, and T.G. Mattson, Linda for Heterogeneous Networks." *Proc. First Annual Workshop on Heterogeneous Processing*, 1992
6. Geist, V.S. Sunderam, Network-Based Concurrent Computing on the PVM System, Concurrency: Practice and Experience, VOL 4, N. 4, PAG: 293-311, 1992
7. J. M. Laarhoven, E.H.L. Aarts, Simulated Annealing: Theory and Applications, Kluver Academic Publishers,
8. Genco, Parallel Simulated Annealing: Getting Super Linear Speedups, *IEEE Proc. of 2nd Euromicro Workshop on Parallel and Distributed Processing*, Malaga (Spain) 1994
9. Glover, F. TABU Search, Techical Report CAAI Report 88-3, Graduate School of Business, University of Colorado, Boulder, Colo, 1988
10. Goldberg, David E. Genetici Algorithms in Search, Optimization and Machine Learning. Reading, Mass,: addison-Wesley, 1989