An evaluation of load-balancing policies on message passing architectures

F. J. Muniz† and E. J. Zaluska‡

† Centro de Desenvolvimento da Tecnologia Nuclear - CDTN/CNEN, C.P. 941, CEP 31120-970, BH, MG, Brazil. E-mail munizfj@urano.cdtn.br

‡ Electronics & Computer Science Department, University of Southampton, SO9 5NH, UK. E-mail ejz@ecs.soton.ac.uk

Abstract

This research investigates dynamic load-balancing mechanisms on a distributed-memory MIMD machine, through the implementation of an application chosen from the computational graphics field (ray-tracing). The application-system performance is compared using user-dependent farmer load-balancing approach and user-transparent, effective and scalable, dynamic load-balancing algorithms. Tasks are generated, at execution time, using a multiple-spawning mechanism based on remote procedural calls. The selected implementation environment is composed of T800 transputers programed in the occam and 'C' languages. The implementation of a real application have evidenced the significance of such load-balancing tools regarded to the practicability of the system-level resource management facilities and performance improvement.

Resumo

Esta pesquisa investiga mecanismos de equalização de demanda computacional (carga) em máquinas MIMD de memória distribuída, através da implementação de uma aplicação escolhida do campo da computação gráfica ('ray-tracing'). O desempenho do sistema é comparado usando-se o mecanismo de equalização de carga 'farmer', não transparente ao usuário, e um mecanismo dinâmico, transparente ao usuário, efetivo e escalável. Processos são gerados, em tempo de execução, usando um mecanismo de 'multiple-spawning' tendo como base primitivas de chamadas de procedimentos remotos. O ambiente de multiprocessamento selecionado é composto de 'transputers' T800, usando linguagens de programação occam e 'C'. A implementação de uma aplicação real evidenciou a significância das ferramentas de equalização no que diz respeito à praticidade das facilidades de gerenciamento de recursos no nível do sistema e o aumento de desempenho.

1 Introduction

This research has been focused on medium grain concurrent computer machines consisting of many entities (nodes or processors) [Fox et al., 1988]. Each entity is a complete computer, built up of CPU, memory and communication facilities. This class of machines, composed of multiple autonomous nodes (computers) interacting through communication devices and operating asynchronously with distinct instruction streams and data sets, is classified as a loosely-coupled distributed-memory MIMD (Multiple Instruction, Multiple Data) machine [Flynn, 1966, Watts, 1989].

Parallel distributed-memory MIMD machines cannot be built by simply interconnecting large quantities of existing CPU, memory and I/O resources, because such an approach could easily produce a system that is unmanageable to program and makes ineffective use of its components. The management of resources is a mechanism used to provide efficient access and use of resources (the multi-processor system) by consumers (any combination of users, which may involve multi-task or multi-thread programes). Some of the objectives of the resource management mechanism are: (1) To decrease the application execution time, thus increasing the overall useful computational performance of the system. (2) To increase portability, facilitate programmability and usability of general purpose multi-processor supercomputer systems.

The main aim of this paper is to compare, through the implementation of an application chosen from the computational graphics field (ray-tracing), the application-system performance using four dynamic-placement load-balancing mechanisms. A short description of this research can be found in the literature [Muniz and Zaluska, 1996].

2 Hardware and software environment

The specific target hardware environment consist of a distributed-memory MIMD Parsys Supernode machine composed of 25 MHz T800 transputers [Nicoud and Tyrrell, 1989, Inmos, 1989]: links were run at 20 Mbits/second (neighbour processors are connected through hard-wired links). Each processor module has 4 Kbytes of internal 1-cycle memory and 4 Mbytes of external 4-cycle memory. The topology selected was a 4×4 torus processor network as shown in Figure 1. The principles demonstrated are general and not restricted to any particular MIMD hardware machine. In terms of programing languages, the research work was developed using the occam 2 language [Inmos, 1988a, Inmos, 1988b, Inmos, 1988c] and the parallel 'C' language [Inmos, 1990].

3 Placement and routing facilities

The design of an efficient, run-time, processes placement system requires a set of 'Remote Procedure Call' (RPC) primitives and creation of communication facilities [Dally and Seitz, 1986, Pountain, 1990, Dally, 1992], to support the exchange of data among the running processes. Autonomous communication mechanism makes intermediate processors transparent to through-routed packets (messages addressed to anyone else). In this research work, fully connected facilities were achieved using a virtual point-to-point store-and-forward message passing mechanism: the automatic routing package 'Virtual Channel Router' (VCR) [Hill, 1993, Debbage, 1993]. The VCR also provides a set of 'Remote Procedure Call' (RPC) primitives - an occam 2 language extension [May, 1989]. Such RPC facilities were used to place run-time processes.

It is necessary to stress that although cost-effective multi-processor systems are now commercially available [Fox et al., 1988, Meiko, 1992], the full utilisation of such parallel hardware power is still restricted because of a lack of facilities such as a proper load-balancing mechanism [Dally et al., 1992]. Any processing system in which the computation and communication demand cannot be predicted *a priori* requires dynamic (run-time) run-time strategies for optimum efficiency.

4 Location policies

The system load demand equalisation can be achieved by a careful placement of newly-created tasks (processes) on under-utilised processors. The system performance is compared using 'Gradient Model' (GM) scheme [Lin and Keller, 1986], the 'Extended Gradient' (EG) mechanism [Muniz, 1994] (which is derived from the GM mechanism) and the farmer approach [Pritchard et al., 1987].

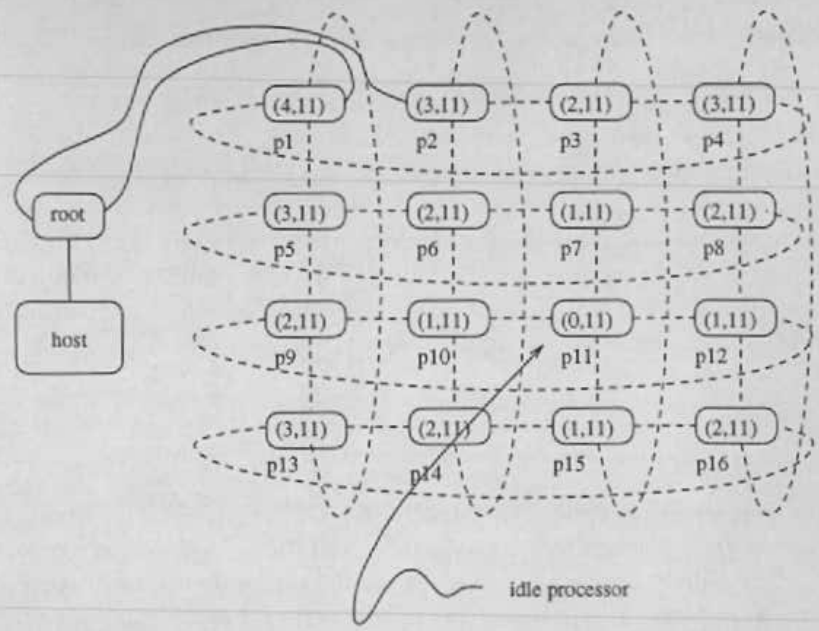
4.1 Gradient model mechanism

In the GM scheme, exchange of load-balancing information is restricted to a small and pre-defined subset of the other nodes in the network, therefore a potential scalable scheme. The status of each node is determined by local processor availability and the status of its neighbour nodes - this status represents the logical distance (number of intermediate neighbour nodes plus one) to the nearest idle node in the network. This information is propagated, from lightly-loaded to over-loaded nodes, through an interchange of messages among neighbour nodes. The load status of each processor, in a network, is updated (and broadcast) at periodic intervals whenever the status of the processor changes. A 'snapshot' of a distributed-memory MIMD transputer network machine, organised as a 4×4 torus topology, using GM scheme is represented in Figure 1.

4.2 Extended gradient algorithm

Although the GM scheme is scalable, it has two serious drawbacks [Nishimura and Kunii, 1990]: (1) Information from idle (lightly-loaded) nodes propagates to over-loaded nodes through intermediate ones. In the worst-case there is a distance of 'd' hops between possible source and destination processors, where 'd' is the maximum distance between any pair of nodes in the network. Since the system load changes dynamically, the processor load status may be considerably out-of-date [Shin and Hou, 1991]. (2) If there are only a few lightly-loaded nodes in the network, more than one source processor may emit a task toward to the same lightly-loaded processors. This 'overflow' effect has the potential to transform lightly-loaded processors into heavily over-loaded ones.

The EG mechanism confirms that a processor is still idle, in order to overcome the problem of out-of-date information and avoid the overflow effect, before transferring load. In other words, once a processing resource is required and not locally available then the local process responsible for resource allocation interrogates the remote processor identified by the GM scheme as lightly-loaded to confirm that it is still available and reserves it to receive the new processing demand. The EG strategy (like the GM scheme) allow processes to be loaded from any processor into the network (i.e. simultaneous loading activity from different user-processes is possible),



Symbols: rectangle with round corners is a processor identified as p_0, p_1, ...; the continuous and dash lines between processors represent the physical link; Tuple (x,y), inside a rectangle with round corners, represents the logical distance (number of intermediate neighbour processors plus one) and the identification of the nearest idle processor. A single idle processor (node 11) is shown.

Figure 1: **Snapshot representation of the GM scheme on a 4×4 torus network.**

providing global placement space. Although communication between any two nodes in the network is possible, this communication facility is used efficiently therefore keeping the EG scalable [Muniz and Zaluska, 1995].

4.3 Farmer distribution approach

The farmer dynamic load-balancing approach is suitable for many scientific applications where the same process is executed in every processor in a network, on different initial data. Once loaded, the master process (the 'farmer') distributes data to identical processes (the 'workers') in different processors. Process placement is decided at compile time and batch of data are distributed on demand to the workers (at run-time). The master process is also in charge of collecting the results generated by the worker processes.

The farmer mechanism requires a good match between the application model and the computational paradigm. It also needs to be tailored to the application, designed and crafted by the user rather than a system-level, user-independent mechanism. Farmer-based approaches make the processing-network-resource-availability visible through just one input processor, the one that executes the farmer load-balancing approach, therefore a centralised scheme. This characteristic of the farmer paradigm causes the bottleneck effect [Lin and Raghavendra, 1992] which makes such approaches not scalable, albeit working reasonably well for systems with a relatively small number of processors (maybe up to tens of processors).

4.4 Control results (farmer + VCR)

An entirely separate implementation of the farmer mechanism was undertaken to act as a control both on the farmer-based mechanism previously described and the EG mechanism version. Although this version also uses the same farmer-based resource management mechanism, it was implemented using the VCR mechanism, instead of using a user-dependent routing mechanism crafted at the user-level. This version thus provides an easier implementation because of the global communication facilities and valency-free mechanism provided by the VCR. Connections between root transputer and each one of the network transputers is through the VCR communication mechanism (the VCR provides, at software-level, global communication facilities for local-connected transputer networks).

5 Processor (CPU) 'busyness' measurement

A simple measure for the instantaneous loading level of a processor (the load status) could be the number of processes that are '**R**eady **t**o **R**un' (RTR). This measure can be correlated with the instantaneous processor utilisation. However, instantaneous sampling of the processor RTR process queue length can produce rapid fluctuations in the instantaneous load of the processors. These fluctuations can be due to processes that may be temporarily queuing for other resources available (such as I/O, timers, etc) or they may also be caused by special system processes of very short life time. The fluctuations may result in misleading information about the load status of a processor, thus resulting in potentially non-productive placement or migration activities. To avoid non-productive placements, it has been suggested [Barak and Shiloh, 1985] that instantaneous processor load is taken every atomic unit of time,

and that it is averaged over a period of time at least of the order of the time required to migrate an average-sized process. Processor idleness was therefore indirectly measured by timing how long a process takes to be scheduled - the implementation is based on a proposal made by Rabagliati [Rabagliati, 1990, Jones et al., 1990]. The process determines the processor idleness by reading the time and then puts itself on the back of the low-priority process queue. When it next executes, it reads the time again and calculates how long it took to get around the processor RTR process queue. In normal operation, the availability measurement process runs concurrently, with other system and application processes. If this is the only process RTR then an 'idle count' variable is incremented. Processor utilisation can then be estimated over a period by comparing the value of the idle count variable with the maximum value expected with no load. A two threshold policy is used. By comparing the idle count variable against these thresholds the processor availability is then classified as 'lightly-loaded' or 'over-loaded'.

6 The application

A real application was chosen from the field of computing graphics (ray-tracing). The ray-tracing application takes as input an array, where each value represents the 'z' coordinate of a surface, and then develops ray-tracing calculations for each possible pixel on the screen. When the processed array is sent to the screen a visual ray-tracing representation of the surface is produced. A surface ray-tracing application is particularly interesting because the input surface size can easily be adjusted and then divided into a desired number of independent subsurfaces (data partitioning), therefore an equal number of totally independent processes can be started at execution time, in order to estimate the efficiency of the dynamic load-balancing mechanism in providing resources 'on demand'. Each one of these run-time-generated processes must compute one of these subsurfaces. A geometric distribution of an application into a number of smaller ones, executing the same code at run-time, is a relatively straightforward method of generating a large number of processes. A system running a larger number of processes than available processors is required for the load-balancing mechanism to manage the system processing capacity effectively. This application was originally designed and implemented in the 'C' programing language to run on a network of transputers hosted by a UNIX workstation [Clarke, 1992]. The application uses 'Xwindows' primitives which are available through a special 'Xtransputer' library and an extended Inmos iserver [Inmos, 1988b], the 'Xiserver' [Clarke, 1992].

6.1 Porting strategy

Some of the main modules of the ray-tracing implementation were ported in their original form with almost no changes: (1) The **xhost** module, executed on the root processor to handle input and output including the 'X-windows' interface. (2) The **xsurf** module, which also executes on the root processor and provides surface generation, loading and collection. (3) The **trace** module, to perform the ray-tracing task. Other modules were modified or eliminated. For example, the code related with the farmer load distribution paradigm is no longer required. Other processes had to be added, for example, the processes related to the real-time process-spawning tree mechanism (the **spawn** module) and a module which is executed in every network processor to handle the surface loading and collection events.

A multiple spawning of processes mechanism is used to distribute the processes at execution time. The actual processing demand is characterised by a single root process and an associated array descriptor of the surface to be computed. The processes are spawned recursively by splitting this descriptor array and passing it down the subtree. At the leaf nodes of the tree (i.e. an array of size one) the actual subsurface is computed. The spawning of processes was achieved using the appropriate VCR RPC primitives [Hill, 1993, Debbage, 1993]. The RPC primitives provide dynamic full occam mechanisms [May, 1989] which enables an occam process to be started at run-time. The **spawn** module, which performs a multiple process-spawning tree mechanism takes as input from the **xhost** module a surface job descriptor and then breaks this into work packets which are passed down to a recursive spawning tree mechanism, such as any divide-and-conquer algorithm. The link between the logical tree node (process) and the physical one (processor) is indicated by the EG mechanism. When a work packet of size one is reached, the subsurface described by this work packet is processed (the ray-tracing calculation is performed). With the return of this process-spawn tree recursive mechanism a result-surface-descriptor is produced, which is than used by the **xhost** module to fetch the processed surface result array.

7 Results

Practical experiments using the ray-tracing application were performed for a screen area of 150×150 pixels which was divided into 64 subsurfaces (hence 64 processes). The total surface area was tuned to provide a total processing demand of about half a minute. Ray-tracing processing activity was performed on this surface using each one of

the load distribution paradigms described previously. For comparison purposes, the farmer strategy using VCR was also run on a network consisting of just a single processor. Average experimental results are shown in Table 1.

C_1	C_2	C_3	C_4	C_5	C_6
	Farmer original, 16 processors	**Farmer** +VCR, 16 processors	**GM** original, 16 processors	**EG** system-level, 16 processors	**Farmer** +VCR, single processor
1	28	26	>50	29	328
2	448	416	>800	464	328
3	73%	79%	<41%	71%	100%

Table 1: **Ray-tracing surface execution time.**

Average measured results for the ray-tracing calculation of a surface array of 150×150 pixels, execution time shown in seconds (line 1). Line 2 represents the machine processing activity available during the experiment in processor-seconds (L_{2i}=number_of_processors×L_{1i}). Line 3 shows the machine utilisation percentage (L_{3i}=100×328/L_{2i}). The result single processor farmer + VCR is presented as control experiment.

8 Conclusions

A benchmark of load-balancing approaches is an intriguing task, since load-balancing policies are optimised for specific objectives and the result varies with different conditions - there is always a particular situation in which a particular load-balancing mechanism is best. It is reasonable to expect that the previous ray-tracing farmer version would have a better performance than that one (farmer + VCR), because internode messages communication was designed and hand-crafted into the user-level code. Surprisingly, the farmer version using VCR had a better performance than the previous (original) custom-designed implementation. Comparison between the performance of the EG mechanism and the farmer approaches has shown an additional cost in terms of the application total CPU-processing time requirements. This increase in CPU-processing demand is mainly due to the load profile demand requirements of the ray-tracing application, a single and intense processing demand burst. Also, a load-balancing mechanism designed specially for a specific application (like farmer) will almost invariably perform better for the particular target than a more general load-balancing mechanism. Another factor which needs to be taken into consideration is that the farmer-base load-balancing approaches are non-scalable and were coded specifically for this ray-tracing application (application dependent), while the EG algorithm is a distributed, simple to use, potentially scalable and application-user-independent. The EG mechanism has produced significant improvements compared to the GM scheme yet retains the considerable advantage of a scalable mechanism. Under these circumstances the performance of the system (machine plus the real application) using EG load-balancing mechanism demonstrated conclusively that the algorithm can provide useful improvements for real applications that generate workload in an unpredictable fashion. In addition, the implementation of a real application using load-balancing mechanism provides a demonstration of usability and applicability of such system-level resource management facilities.

Enhancing the basic features provided by a concurrent architecture will probably become a major focus of computer engineering and architecture research [Inmos, 1991, Dally et al., 1992]. Such improved architectures will support message-passing parallel computation directly in a cost-effective way. They will incorporate facilities optimised to reduce practical design constraints, including facilities currently implemented in software such as remote procedure calls, automatic processor 'busyness'-level measurement, exchange of load level among processors and hardware communication devices. Such machines will facilitate the exploitation of load-balancing mechanisms using significantly finer-grain programing models than those practical at present.

Finally, the 'ease of use' of a parallel system is of vital importance if these machines are to realise their potential and become widely adopted by end users. The provision of automatic load-balancing at the system-level makes an important contribution to this ease of use and this aspect has also been a major aim of the work reported in this research.

9 Acknowledgements

We wish to thank the Brazilian organisations CNPq, CNEN and FAPEMIG for the financial support given to FJM.

References

[Barak and Shiloh, 1985] Barak, A. and Shiloh, A. (1985). A distributed load-balancing policy for a multicomputer. *Software-practice and Experience*, 15(9):901–913.

[Clarke, 1992] Clarke, N. (1992). Private communication.

[Dally, 1992] Dally, W. J. (1992). Virtual channel flow control. *IEEE Trans. on Parallel and Distributed Systems*, 3(2):194–205.

[Dally et al., 1992] Dally, W. J., Fiske, J. A. S., Keen, J. S., Lethin, R. A., Noakes, M. D., Nuth, P. R., Davison, R. E., and Fyler, G. A. (1992). The message-driven processor: a multicomputer processing node with efficient mechanisms. *IEEE Micro*, 12(2):23–39.

[Dally and Seitz, 1986] Dally, W. J. and Seitz, C. L. (1986). The torus routing chip. Technical report, California Institute of Technology, 5208:TR:86. To be published in Journal of Distributed Computing, 1(3), 18 pages.

[Debbage, 1993] Debbage, M. (1993). *Reliable communication protocols for high performance computing.* PhD thesis, University of Southampton.

[Flynn, 1966] Flynn, M. J. (1966). Very high-speed computing systems. In *Proc. of the IEEE*, pages 1901–1909.

[Fox et al., 1988] Fox, G. C., Johnson, M. A., Lyzenga, G. A., Otto, S. W., Salmon, J. K., and Walker, D. W. (1988). *Solving Problems on Concurrent Processors*, volume I. Prentice-Hall International.

[Hill, 1993] Hill, M. B. (1993). *Software environments for parallel computing.* PhD thesis, University of Southampton.

[Inmos, 1988a] Inmos (1988a). *occam 2 Reference manual.* Prentice-Hall International.

[Inmos, 1988b] Inmos (1988b). *occam 2 Toolset manual (D705).* Bristol, United Kingdom.

[Inmos, 1988c] Inmos (1988c). *Transputer instruction set: a compiler writer's guide.* Prentice-Hall International.

[Inmos, 1989] Inmos (1989). *The Transputer databook.* Bath Press Ltd.

[Inmos, 1990] Inmos (1990). *Parallel C toolset.*

[Inmos, 1991] Inmos (1991). *The T9000 transputer products overview manual.* Prentice-Hall International.

[Jones et al., 1990] Jones, G., Zeppenfeld, K., and Goldsmith, M. (1990). Measuring the busyness of a transputer. *occam user group newsletter*, (12):57–64.

[Lin and Keller, 1986] Lin, F. C. H. and Keller, M. R. (1986). Gradient model: a demand-driven load balancing scheme. In *IEEE Conf. on Distributed Systems*, pages 329–336.

[Lin and Raghavendra, 1992] Lin, H. C. and Raghavendra, C. S. (1992). A dynamic load-balancing policy with a central job dispatcher (LBC). *IEEE Trans. on Software Engineering*, 18(2):148–158.

[May, 1989] May, D. (1989). Implementing full occam. ESPRIT Proj. 2701: PUMA, Inmos.

[Meiko, 1992] Meiko (1992). Information & product news for Meiko users. Technical report, Meiko. 39 pages.

[Muniz, 1994] Muniz, F. J. (1994). *Parallel load-balancing on message passing architectures.* PhD thesis, University of Southampton. Viva on March, 18^{th}.

[Muniz and Zaluska, 1995] Muniz, F. J. and Zaluska, E. J. (1995). Parallel load-balancing: An extension to the gradient model. *Parallel Computing*, 21(2):287–301.

[Muniz and Zaluska, 1996] Muniz, F. J. and Zaluska, E. J. (1996). Parallel ray-tracing on MIMD machine using dynamic load-balancing mechanisms. In Pereira, C. E., editor, *Real-Time Programming*, Porto Alegre, RS, Brasil. Universidade Federal do Rio Grande do Sul - UFRGS. To be published by Elsevier Press.

[Nicoud and Tyrrell, 1989] Nicoud, J. D. and Tyrrell, A. M. (1989). The transputer T414 instruction set. *IEEE Micro*, pages 60–75.

[Nishimura and Kunii, 1990] Nishimura, S. and Kunii, T. L. (1990). A decentralized dynamic scheduling scheme for transputers networks. In Kunii, T. L. and May, D., editors, *Proc. of the 3^{rd} Transputer/occam Int. Conf.*, pages 181–194, Tokyo, Japan.

[Pountain, 1990] Pountain, D. (1990). Virtual channels: The next generation of transputers. *BYTE*, pages 3–12, European supplement. April.

[Pritchard et al., 1987] Pritchard, D. J., Askew, C. R., Carpenter, D. B., Glendinning, I., Hey, A. J. G., and Nicole, D. A. (1987). Practical parallelism using transputer arrays. In Goos, G. and Hartmanis, J., editors, *Lecture Notes in Computer Science*, pages 278–294. Springer-Verlag.

[Rabagliati, 1990] Rabagliati, A. (1990). Private communication.

[Shin and Hou, 1991] Shin, K. G. and Hou, C.-J. (1991). Effective load sharing in distributed real-time systems. In *3^{d} IEEE Symp. on Parallel and Distributed Processing*, pages 670–677, Dallas, Texas.

[Watts, 1989] Watts, S. (1989). Parallel tracks to standard processing. *New Scientist*, 123(1677):44–47.