Self-Tuned Parallel Processing System for Heterogeneous Clusters

J. Barbosa J. Tavares
A. Padilha
DEEC, Faculdade de Engenharia da Universidade do Porto
Rua Dr. Roberto Frias, 4200-465 Porto, Portugal

Abstract ***Automatic program execution control is a demanding tool for heterogeneous environments, due to the variable availability of processors and network bandwidth. Issues such as load distribution have to be re-analyzed since strategies developed for homogeneous machines fail to optimize the computation time in heterogeneous networks. The final user should not be involved in deciding which machines should be selected, the data distribution policies that best fit each algorithm, etc. The present work presents such a tool, designed for interactive processing in a distributed computing environment, whose main objective is to minimize the applications computation time. Results are presented for an object tracking algorithm, either in homogeneous and heterogeneous networks showing the advantage of automatic control.***

Keywords: personal computer cluster, automatic scheduling, linear algebra, object tracking, execution time minimization

1 Introduction

A strong motivation for parallel processing in interactive applications comes from image and image sequence analysis needs posed by various application domains, which are becoming increasingly more demanding in terms of the detail and variety of the expected analytic results, requiring the use of more sophisticated image and object models, such as physically-based deformable models [16, 19].

The approach adopted was to develop parallel software to be executed, in a distributed manner, by the machines available in an existing computer network, taking advantage of the well known fact that many of the computers are often idle for long periods of time [17]. The automatic control tool implements the appropriate managing software in order to put idle computers at the service of the more computationally demanding applications.

Distributed processing is frequently divided in two classes: High Throughput Computing (HTC) and High Performance Computing (HPC). HTC systems are characterized by the amount of floating point operations they can execute in a given period of time, for example, a day or a week. These machines are built by general purpose computer networks, already available and used for sharing peripherals, file systems and other resources. The use of this kind of computational environment for parallel processing is based on the utilization of processor cycles that would be otherwise lost, thus maximizing the use of the processing capacity installed. Since the network is not designed for parallel processing it imposes performance limitations, although they can be used efficiently to execute sequential tasks in parallel or parallel tasks of high granularity. Examples of HTC systems are Condor [17], Batrun [20] and Hector [18].

HPC systems are machines built by complete computers, such as workstations, connected by a high bandwidth network. Examples are the NOW [1] and HPVM [7] systems. The aim of these systems is to achieve a performance level that can be compared to some supercomputers, with the benefit of lower machine acquisition and maintenance costs.

An important difference between HTC and HPC systems is that in the former it is usually implemented load sharing instead of load balancing; the first assigns tasks to available processors without consideration of the processing time, while in the second the aim is to assign the same amount of work to each processor (a characteristic of HPC systems). In [2] and [3] a survey of these two types of systems is presented.

The system presented in this work does not nicely fit into any of the classes mentioned, however, it is closer to the characteristics of HTC systems. The aim is to execute interactive applications in an existing local computer network, in order to reduce the response time through the use of the available processing capacity.

2 Computational environment and algorithms

The target computational environment are existing group clusters formed by desktop computers. These clusters are characterized by having a low cost interconnection network, such as a 10/100 *Mbits* Ethernet, connecting different types of processors, of variable processing capacity and amount of memory, thus forming a heterogeneous parallel virtual computer.

Although several computational models [10, 15, 21] have been proposed to estimate the processing time of a parallel algorithm in a distributed memory machine, considering the target machine, a simplified model was proposed in [5]. This model accounts separately for the communication and computation time used by an algorithm. The time to send a message (T_C) of nb bytes is given by

$$T_C(nb) = T_L K + \frac{nb}{BW} \tag{1}$$

where K is the number of packets and BW the network bandwidth. The parallel component T_P of the computational model represents the operations that can be divided over a set of p processors obtaining a speedup of p, i.e. operations without any sequential part:

$$T_P(n,p) = \frac{\psi(n)}{\sum_{i=1}^{p} S_i} \tag{2}$$

The numerator $\psi(n)$ is the cost function of the algorithm measured in floating point operations ($flop$) as a function of the problem size n. For example, to multiply square matrices of size n, the cost is $\psi(n) = 2n^3$ [11].

Each algorithm or task is decomposed in indivisible operators for which parallel code exists. When a parallel algorithm is launched in one machine (the master process), the work is scheduled according to the available processors and choosing, for each operation, the processor grid that optimizes its execution time [4], allowing data redistribution if the optimal grid changes from operation to operation.

The operators considered in this work are a set of basic linear algebra operations that are applied in the field of image analysis, e.g. for object modeling. They are the right looking variant of LU factorization [12], tridiagonal reduction of symmetric matrices [9], QR iteration [13] and matrix correlation.

The operators were rewritten in order to consider heterogeneous environments whose aim is to assign the same amount of processing time instead of the same amount of work, as for homogeneous machines. In particular, the QR iteration was implemented by a parallelization strategy different from the one proposed in [8], adapted to the machine. It reduces the communications among slave processors to zero, and it keeps only communications from the master process to the slaves.

3 Parallel processing system

The system is a self-tuned tool that automatically selects processors, data distribution strategies and processor layout that minimize the processing time of a given application, based on the amount of data required to communicate and process in each operation. Figure 1 shows the system configuration.

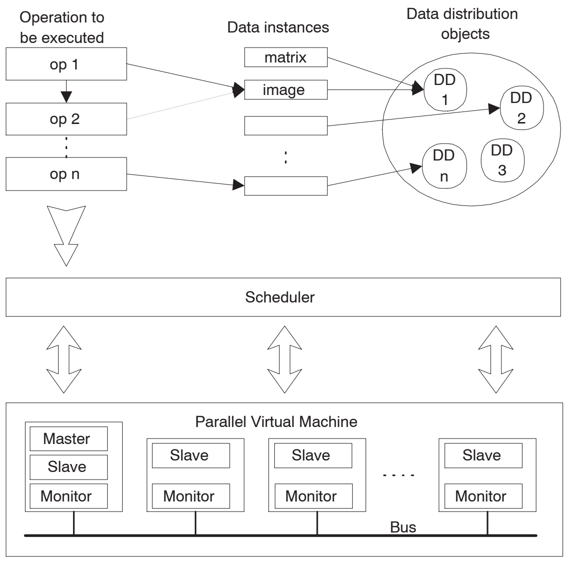


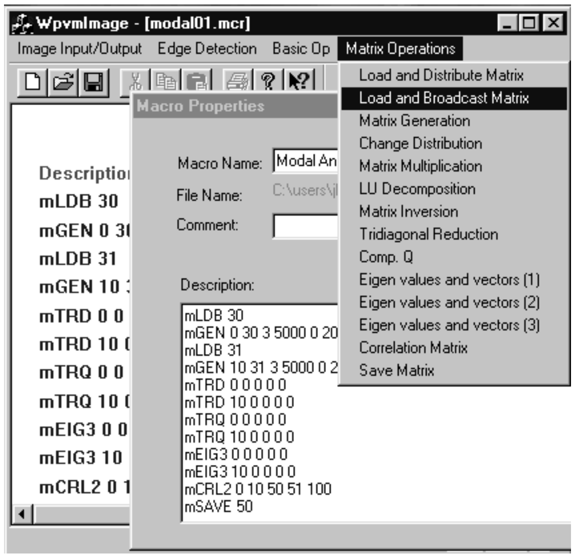
Figure 1: Interaction of system components

The system is divided in three main blocks: scheduler, distributed processing management (DPM) and user interface. The scheduler is implemented by one object created in the master process and it has the function of deciding, for each operator, the processor layout that minimizes its execution time. Each operator codes in its definition class the amount of work to be done and the required communications, referred to the amount of data, which are the parameters for the computational model. The scheduler output is the complete algorithm to execute which has the additional information of the processor grid and data layout to use in each operator and, if necessary, the redistribution operations introduced between operators.

The DPM, implemented in the slave and monitor processes, uses the scheduler output to create simultaneously in every participating processor the data distribution objects required to allocate and address the data objects in each processor according to the amount of work assigned. The data blocks assigned to each processor are determined by the data distribution algorithms (linear, cyclic, group[1] [6] which are coded in the data distribution class.

The data objects are distributed over the processors that participate in the computation. In each one there is a copy of the data distribution object to control the data access. The DPM has a set of instructions to manage the parallel machine, such as packing and broadcasting data buffers, creation and release of data structures, processor grid control, process synchronization and collection of processor state information.

The user interface is implemented in two levels: algorithm definition (macros) and execution. To define an algorithm the user has to identify which operators are involved and describe them as a sequence of operations. The data buffers are indexed by numbers, and to use the result of one operation in another, only those numbers have to be indicated. Figure 2 shows the interface for macro definition. There is a set of instructions available for input/output, and only the buffers mentioned in an output instruction will be saved to the disk or shown in the display.



Macro Name: Modal An
File Name: C:\users\jl
Comment:
Description:

```
mLDB 30
mGEN 0 30 3 5000 0 20
mLDB 31
mGEN 10 31 3 5000 0 2
mTRD 0 0 0 0 0
mTRD 10 0 0 0 0
mTRQ 0 0 0 0 0
mTRQ 10 0 0 0 0
mEIG3 0 0 0 0 0
mEIG3 10 0 0 0 0
mCRL2 0 10 50 51 100
mSAVE 50
```

Figure 2: Macro definition interface

[1]The group block distribution has been proposed to improve the processing time of inherently sequential algorithms such as the linear algebra operators in heterogeneous clusters

The macro execution interface allows a non-expert user to execute algorithms defined previously. After opening a macro, the user only has to choose the run command, which will invoke a dialog to ask for the input and output files, as mentioned in the macro by load, input, save and output instructions.

4 Results

The results presented next demonstrate the applicability of the parallel processing system in an image analysis algorithm, namely the modal matching algorithm [19]. The aim is to model the deformation of an object in an image sequence as shown in figure 3, in order to obtain the correspondence between object points.

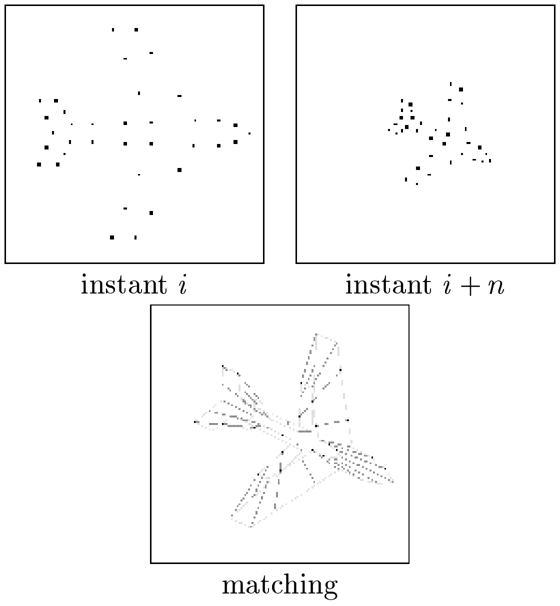


Figure 3: Modal matching algorithm

This is a simple example where deformation occurs only in rotation and scale, however, the method can be applied for shape deformation. From the set of points describing each object (instants i and $i+n$), the algorithm creates two symmetric matrices representing the objects and performs finite element analysis. The operators that compose the algorithm are eigenvector computation and matrix correlation. The first is further subdivided into three operations: tridiagonalization, orthogonalization and QR iteration.

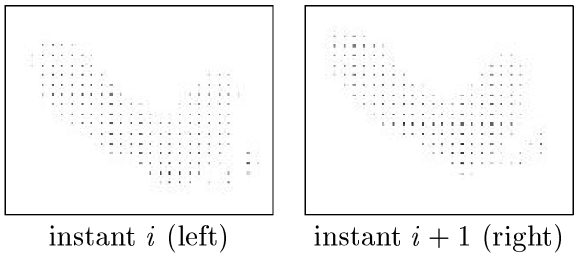


Figure 4: Footprint data acquisition; pixel intensity expresses pressure

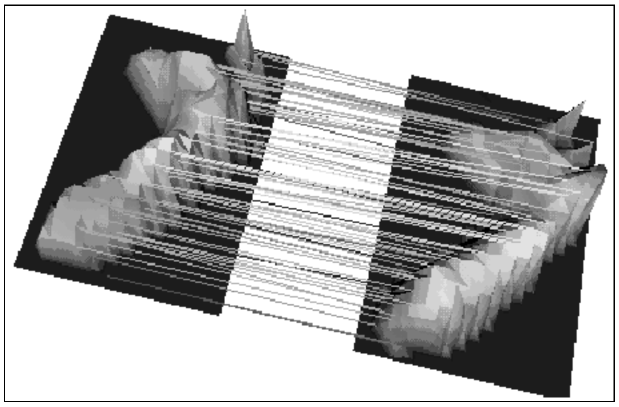


Figure 5: Modal matching algorithm in pedobarography

Figure 4 shows two segmented images of a footprint, in normal walking conditions, where the pixel intensity represents the pressure made on the acquisition platform, which is the relevant information for the clinical analysis. In this case there is a significant shape deformation. The set of points represents a 2D surface with an intensity value per point that is mapped into the third dimension. Therefore, a 3D representation can be obtained, and the matching is computed for this representation (figure 5). An object of n points is represented by a $(3n, 3n)$ matrix, thus increasing considerably the problem size as compared to the 2D analysis.

Table 1 presents results for a machine composed by {244, 244, 161, 161, 60, 50, 49} $Mflops$ processors, connected by a 100 $Mbit$ Ethernet, in matching the objects of figure 4,

Operator	Processor grid	
Object	A	B
TRD	(1,3)	(1,3)
ORT	(1,6)	(1,6)
QR	(6,1)	(6,1)
MC	(1,6)	
T_1/T_{VM}	1621.7/615.8	
$Speedup$	2.63	
E_H	69.9 %	

Table 1: Results for heterogeneous machine; A:331 and B:329 points

where the left and right objects are described by 331 and 329 points, respectively.

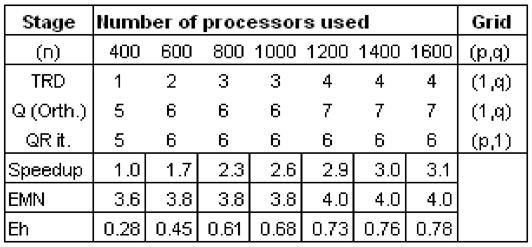
The processor grid changes between operators by data redistribution operations, whose time is included in the virtual machine processing time (T_{VM}). The speedup achieved is not high, considering that six processors are used in the computation. However, for a heterogeneous machine this performance measure is ambiguous since it does not reflect the increase of processing capacity compared to the sequential processing. Therefore, efficiency is computed based on the equivalent machine number:

$$EMN(p) = \sum_{i=1}^{p} \frac{S_i}{S_1} \tag{3}$$

where S_1 is the capacity of the processor that executes the sequential version (the fastest in our tests) [5]. Heterogeneous efficiency ($E_H(p)$), using p processors, is computed as:

$$E_H(p) = \frac{speedup}{EMN(p)} \tag{4}$$

For this example $EMN(6) = 3.77$, obtaining $E_H = 69.9\%$. Efficiency measures the quality of parallelization by giving a measure of the time spent in computation over the total time. To assess the machine the eigenvector algorithm was run individually, which corresponds to the first stage of the computation of the modal analysis algorithm, operations (TRD, ORT, and QR) of table 1, for several matrices.



Stage	Number of processors used							Grid
(n)	400	600	800	1000	1200	1400	1600	(p,q)
TRD	1	2	3	3	4	4	4	(1,q)
Q (Orth.)	5	6	6	6	7	7	7	(1,q)
QR it.	5	6	6	6	6	6	6	(p,1)
Speedup	1.0	1.7	2.3	2.6	2.9	3.0	3.1	
EMN	3.6	3.8	3.8	3.8	4.0	4.0	4.0	
Eh	0.28	0.45	0.61	0.68	0.73	0.76	0.78	

Table 2: Results for eigenvector computation

Table 2 shows the results of speedup and heterogeneous efficiency; the processor grid used depends on the matrix and the operator. It can be observed that for matrices over 600^2 elements the heterogeneous efficiency is over 45%. For 800^2 matrices $E_H = 61\%$ with 3.8 equivalent processors. In [14] it was measured an efficiency of 57.9% with 4 vector processors of the distributed memory computer Fujitsu VPP300, in the computation of the eigenvector algorithm, using the same algorithm and a 802^2 element matrix.

In terms of parallelization quality the results obtained for the virtual machine, when using the automatic processing control tool, are in the same range of values obtained with dedicated machines. However, the computation times are naturally very different: 1.783 and 97.7 seconds for Fujitsu and our virtual machine, respectively.

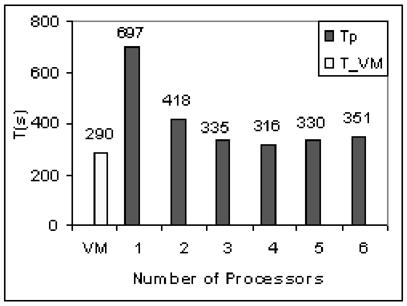


Figure 6: Modal analysis results in a homogeneous machine

The automatic tool presented here can be useful also for homogeneous machines, due to the scheduler capacity. As shown in the former example, the processor grid can change,

Operator	Processor grid	
Object	A	B
TRD	(1,2)	(1,2)
ORT	(1,6)	(1,5)
QR	(5,1)	(4,1)
MC	(1,6)	
T_1/T_{VM}	697/290	
Speedup	2.40	
Efficiency	40.0 %	

Table 3: Results for homogeneous machine; A:251 and B:174 points

due to different operator scalabilities, which will also be true for homogeneous machines. Figure 6 compares the processing time, for two other objects of 251 and 154 data points (not shown), obtained using the scheduler (T_{VM}) and the time obtained using the same number of processors all over the algorithm, from 1 to 6. The machine is formed by 6 processors of 140 $Mflops$ and a 10 $Mbits$ Ethernet. Without processing time estimation, an user having 6 processors available, probably would use all of them from the beginning, although the minimum time is obtained with 4 processors. However, this is higher than the time obtained with the automatic tool (T_{VM}). Table 3 shows the results for the homogeneous machine.

5 Conclusions

An automatic high-level parallel processing tool was presented in this work. It allows a non-expert user to run pre-defined algorithms, referred to as macros. At each run the tool selects, from the pool of available processors, those for which the estimation based on the computational model guarantees the minimum processing time, scheduling work in proportion to their processing capacity. It was shown that the tool reduces the waiting time in interactive applications, achieving high efficiency both in homogeneous and in heterogeneous machines.

Speedup results presented in tables 1, 2 and 3, are lower bounds, since the parallel processing time (T_{VM}) is compared to the sequential time measured in the fastest machine available on the network. Higher speedup values would be obtained if any other machine was chosen, which is also a valid option since the parallel job can be submitted from any computer. From the user point of view the option is to execute locally (any single computer) or in parallel. The sequential time is estimated for the computer that submits the work and the parallel processing time is constant as long as the same computers are available.

In conclusion, the use of the automatic control tool permits an efficient utilization of existing computer networks allowing that interactive applications use more sophisticated algorithms without requiring an investment in equipment. Nevertheless, the upgrade or acquisition of new equipment benefits all users independently of who submits the job.

References

[1] T. Anderson, D. Culler, D. Patterson, and The `NOW` Team. A case for NOW (Network of Workstations). *IEEE Micro*, (2):54–64, February 1995.

[2] M. Antonioletti. Load sharing across networked computers. Technical report, The University of Edinburgh, http://www.epcc.ed.ac.uk/epcc-tec/-documents/tw-load/, December 1997.

[3] M. Baker, G. Fox, and H. Yau. Cluster computing review. Technical report, Syracuse University, http://www.npac.-syr.edu/techreports/hypertext/sccs-0748/, November 1995.

[4] J. Barbosa and A.J. Padilha. Algorithm-dependent method to determine the optimal number of computers in parallel virtual machines. In *VECPAR'98, 3rd International Meeting on Vector and Parallel Processing (Systems and Applications)*, volume 1573. Springer-Verlag LNCS, 1998.

[5] J. Barbosa, J. Tavares, and A. J. Padilha. Linear algebra algorithms in a heterogeneous cluster of personal computers. In *Proceedings of 9th Heterogeneous Computing Workshop*, pages 147–159. IEEE CS Press, May 2000.

[6] J. Barbosa, J. Tavares, and A.J. Padilha. A group block distribution strategy for a heterogeneous machine. In *Submitted to Euro-Par 2001.*

[7] A. Chien, M. Lauria, R. Pennington, M. Showerman, G. Iannello, M. Buchanan, K. Connelly, L. Giannini, G. Koenig, S. Krishnamurthy, Q. Liu, S. Pakin, and G. Sampemane. Design and evaluation of an HPVM-based Windows NT Supercomputer. *The International Journal of High-Performance Computing Applications*, 13(3):201–219, Fall 1999.

[8] J. Choi, J. Dongarra, L. S. Ostrouchov, A. P. Petitet, D. W. Walker, and R. C. Whaley. The design and implementation of the ScaLAPACK LU, QR, and CHOLESKY factorization routines. *Scientific Programming*, 5:173–184, 1996.

[9] J. Choi, J. Dongarra, and D. Walker. The design of parallel dense linear software library: Reduction to hessenberg, tridiagonal and bidiagonal form. Technical Report LAPACK Working Note 92, University of Tennessee, Knoxville, January 1995.

[10] D. Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. Logp: Towards a realistic model of parallel computation. In *4 ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, San Diego, CA, 1993.

[11] J. W. Demmel. *Applied Numerical Linear Algebra.* SIAM, 1997.

[12] J. Dongarra, Sven Hammarling, and David W. Walker. Key concepts for parallel out-of-core lu factorization. Technical Report CS-96-324, LAPACK Working Note 110, University of Tennessee Computer Science, Knoxville, April 1996.

[13] Gene Golub. *Matrix Computations.* The Johns Hopkins University Press, 1996.

[14] D. L. Harrar and M. R. Osborne. Solving large-scale eigenvalue problems on vector parallel processors. In *VECPAR'98, 3rd International Meeting on Vector and Parallel Processing (Systems and Applications)*, volume 1573, pages 100–113. Springer-Verlag LNCS, 1998.

[15] J.F. JáJá and K.W. Ryu. The block distributed memory model. Technical Report CS-TR-3207, University of Maryland, January 1994.

[16] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, pages 321–331, 1988.

[17] M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *Proceedings of IEEE International Conference on Distributing Systems*, pages 104–111, 1988.

[18] S. Russ, J. Robinson, B. Flachs, and B. Heckel. The hector distributed runtime environment. *IEEE Trans. on Parallel and Distributed Systems*, 9(11):1102–1114, November 1998.

[19] L. Shapiro and J. M. Brady. Feature-based correspondence: an eigenvector approach. *Butterworth-Heinemann Lda*, 10(5), June 1992.

[20] F. Tandiary, S. C. Kothari, A. Dixit, and W. Anderson. Batrun: Utilizing idle workstations for large-scale computing. *IEEE Parallel and Distributed Technology*, pages 41–48, Summer 1996.

[21] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, August 1990.