Parallel Image Processing System on a Cluster of Personal Computers

(Best Student Paper Award: First Prize)

J. Barbosa*, J. Tavares**, and A.J. Padilha

FEUP-INEB, Rua Dr. Roberto Frias, 4200-465 Porto (P)
e-mail: jbarbosa@fe.up.pt

Abstract. The most demanding image processing applications require real time processing, often using special purpose hardware. The work herein presented refers to the application of cluster computing for off line image processing, where the end user benefits from the operation of otherwise idle processors in the local LAN. The virtual parallel computer is composed by off-the-shelf personal computers connected by a low cost network, such as a 10 *Mbits/s* Ethernet. The aim is to minimise the processing time of a high level image processing package. The system developed to manage the parallel execution is described and some results obtained for the parallelisation of high level image processing algorithms are discussed, namely for active contour and modal analysis methods which require the computation of the eigenvectors of a symmetric matrix.

1 Introduction

Image processing applications can be very computationally demanding due to the large amount of data to process, to the response time required, or to the complexity of the image processing algorithms. A wide range of general purpose or custom hardware has been used for image processing. SIMD computers, using data parallelism, are suitable for low level image analysis, where each processor performs a uniform set of operations based on the image data matrix in a fixed amount of time; in [28] a special purpose SIMD computer with 1024 processors was presented. Systolic Arrays [11] which can exploit the regular and constant-time operations of an algorithm are also an option.

MIMD computers, commonly used in simulation, are suitable for high level image processing, such as pattern recognition, where each processor is assigned an independent operation [3]. For real time vision applications special MIMD computers were developed; e.g. ASSET-2 based on PowerPC processors for computation and on Transputers for communication [29]. MIMD computers were characterised by exploiting a variety of structures; however, technological factors have been forcing a convergence towards systems formed by a collection of

* PhD grant BD/2850/94 PRAXIS XXI

** PhD grant BD/3243/94 PRAXIS XXI

essentially complete computers connected by a communications network [9]. The processors in these computers are the same ones used in current workstations. Therefore, the idea of forming a parallel computer from a collection of off-the-shelf computers comes naturally, and fast communication techniques were also developed for that purpose [25]. Several cluster computing systems have been developed, e.g. the NOW project [2].

Our aim is not to build a specific cluster of personal computers for parallel image processing, but rather to perform parallel processing on already existing group clusters, where each node is a desktop computer running the Windows operating system. These clusters are characterised by having a low cost interconnection network, such as a 10/100 *Mbits/s* Ethernet, connecting different types of processors, of variable processing capacity and amount of memory, thus forming a heterogeneous parallel virtual computer. Due to network restrictions, which do not allow simultaneous communication among several nodes, the application domain is restricted to about one or two dozens of processors.

The motivation for a parallel implementation of image algorithms 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 (e.g., physically-based deformable models), and of more complex algorithms, while the timing constraints are kept very stringent.

A promising approach to deal with the above requirements consists in developing 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 [20]. It is quite common in many organisations that a standard network connects several general purpose workstations and personal computers, accumulating a very substantial computing power that, through the use of appropriate managing software, could be put at the service of the more computationally demanding applications.

Existing software, such as the Windows Parallel Virtual Machine (WPVM) [1], allows building parallel virtual computers by integrating in a common processing environment a set of distinct machines (nodes) connected to the network. Although the parallel virtual computer nodes and the underlying communication network were not designed for optimised parallel operation, very significant performance gains can be attained if the parallel application software is conceived for that specific environment.

2 Image Algorithms and Systems

The image algorithms that have been parallelised consist of a set of low level image processing operations namely edge detection [27,6], distance transform, convolution mask, histogramming and thresholding, whose suitability to the cluster architecture was analysed in [4]. A set of linear algebra algorithms which are building blocks for many high level image processing methods was also im-

plemented. These algorithms are the matrix product [14], LU factorisation [7], tridiagonal reduction [8], symmetric QR iteration [15], matrix inversion [23] and matrix correlation.

In this paper, the results presented focus on high level image processing algorithms, namely active contours [19] and modal analysis [26].

Some image processing systems have been proposed to run on a cluster of personal computers. In [17] two highly demanding vision algorithms were tested giving superlinear speedup, due to memory pagination on a single workstation. The machines formed an homogeneous computer. In [18] a high level interface parallel image processing library is presented and results are reported for low level image operations on an Ethernet network of HP9000/715 workstations and on an ATM network of SGI workstations. In [21] a machine independent methodology was proposed for homogeneous computers; results were presented separately for two SMP workstations with two and eight processors, not requiring communication between machines.

Our implementation differs from the ones mentioned above as it considers a general bus type heterogeneous cluster where data is distributed in order to obtain a correct load balancing, and also because the number of processors that participate in a distributed algorithm vary dynamically in order to minimise the processing time of each operation [5].

3 System Architecture

The computers that belong to the virtual machine run a process to monitor the percentage of processor time spent with the local user. Conceptually, local users have priority over the distributed application and the computer will not be available if the mean local user time is above a minimum threshold during a specified period of time, e.g. 5 seconds.

Each algorithm or task is decomposed until indivisible operations are obtained to which parallel code exists. When a parallel algorithm is launched the master process schedules work to the processors of the virtual machine according to their availability and choosing a number of processors that minimise the processing time of individual operations, allowing data redistribution if the optimal grid [4] of processors changes from operation to operation.

As an example, the algorithm to extract the contour of an object can be decomposed into edge enhancement, thresholding and contour tracking operations.

3.1 Hardware Organisation and Computational Model

The hardware organisation is shown in figure 1. Each node of the virtual machine is a personal computer under the Windows NT operating system, running WPVM software to communicate. The interconnection network is an Ethernet at 10/100 *Mbits/s*.

Several computational models [9, 30, 16] were proposed in order to estimate the processing time of a parallel program in a distributed memory machine.

Fig. 1. Hardware organisation

Although they could be adapted for the cluster of personal computers, a specific and simplified model is presented below.

The total processing time is obtained by summing the time spent in communications (T_C) and the parallel processing time (T_P). Each node of the machine is characterised by the processor capacity S_i , measured in $Mflop/s$. The network is characterised by allowing only one message to be broadcast at a given time, the latency time (T_L) and the bandwidth (LB). The time to send a message (T_C) composed by nb bytes is given by:

$$T_C = T_L K + \frac{nb}{LB}, \quad K = \left\lceil \frac{nb}{packetsize} \right\rceil \quad (1)$$

The value K multiplies T_L due to the partition of each message into packets of length 46 to 1500 bytes ($packetsize$), existing a latency time for each packet; 1024 is a typical packet size.

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:

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

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 $\psi(n) = 2n^3$ [10].

3.2 Software Organisation

Each operation is represented by an object containing the parallel and serial implementation of the code, since the system can schedule a sequential execution remotely if it is advantageous. The object associated to the operation also contains information on the computational complexity and the amount of data required to exchange in order to complete the operation. Based on these parameters the system determines the number and the identities of the intervening processors, in order to minimise the operation processing time [4].

Each data instance to be processed, either an image or a matrix, is represented by an object responsible for accessing data items correctly according to the data distribution information.

Data distribution is represented by independent objects with functions to locate any item of data and to translate global to local indexes and vice-versa.

Each object can be shared by more than one data instance. Figure 2 shows the software organisation.

Fig. 2. Software organisation

The user describes a macro of sequential operations to be executed referring the data instances to be processed. The system executes each operation in parallel determining for each one the number of processors to be used in order to minimise the processing time. The data distribution suitable for each operation is coded in the operation code.

```
Input i1 image1.bmp
Shencastan i1 i2 i3 0
Histogram i2 outfile.txt i4
Output i2
Output i4
```

Fig. 3. Macro describing the operations to be executed

Figure 3 shows an example of a macro. The input file *i1* is subject to an edge detector [27]; the operator outputs, the gradient's magnitude and direction, are stored in *i2* and *i3* respectively. The histogram is then computed and displayed as an image, its data being also saved in a text file.

3.3 Data Distribution and Load Balancing

Different strategies are applied to images and matrices. Images are partitioned in blocks of contiguous rows or columns and the blocks are assigned to each process [4]. This distribution is suitable for data independent image operators. Matrices are organised in square blocks of data and a novel version [5] of the block cyclic domain distribution [13], adapted to an heterogeneous environment, is used for assigning them to the processor grid.

A balanced distribution is achieved by a static load distribution made prior to the execution of the parallel operation. To achieve a balanced distribution in the

heterogeneous machine the relative amount of data assigned to each processor, l_i , is a function of its processing capacity compared to the entire machine:

$$l_i = S_i / \sum_{k=1}^p S_k \quad (3)$$

For matrices, due to block indivisibility, it is not always possible to ensure an optimal load balancing; however, the scheduler computes the optimal solution for a given network [5]. The processor placement on the virtual grid is also done in order to achieve a balanced distribution.

4 Parallel Implementation of the Active Contour Algorithm

An active contour is defined as an energy minimising curve subject to the action of internal forces and influenced by external image forces which move the contour to the relevant features in the image, such as lines and edges [19].

Active contours can be used for a variety of feature extraction operations in images, such as detection of lines and edges, detection of subjective contours, tracking analysis in a sequence of images or correspondence analysis in stereo images.

Fig. 4. Application of the active contour algorithm in an angiocardiographic image

Figure 4 (rightmost image) shows the contour detection over the original image of 64 *KB*. From an initial position (arbitrarily or interactively defined) and by using an iterative process, the contour moves in order to minimise its energy. The final position corresponds to a local minimum of the energy function defined. In this position, all the forces applied to the contour are mutually cancelled, so that the contour does not move. The energy function was computed based on the edge detection map (leftmost image) and the distance transform map (middle image). The quality of the detection depends on these two images. Different energy functions can be used [24]; however, not all are suitable for every application.

The active contour points which are distant from the edges are pushed in their direction by the distance transform. The points close to edges are mostly influenced by the edge map energy which locally refines the detection.

Fig. 5. Active contour algorithm decomposed in indivisible tasks

Figure 5 shows the tasks required to apply the active contour algorithm. The computation methodology is to sequentially execute each parallelised task, choosing the grid of processors that minimises the individual processing time and, consequently, the overall time.

The image operators have been discussed in another paper [4]. Therefore, only the parallelisation of the LU factorisation routine is considered here.

4.1 LU Factorisation Algorithm

The LU factorisation algorithm is applied in order to solve directly the system of equations resulting from the active contour internal forces: elasticity and flexibility. The implementation follows the right-looking variant of the algorithm proposed in [12]. However, adaptations were made at the load distribution level in order to obtain a balanced load for heterogeneous machines. Figure 6 (left) shows the load distribution obtained for a heterogeneous virtual machine.

Fig. 6. LU and QR load distribution for a matrix size of 1800 and 1600 respectively for the machine $M=\{244, 244, 161, 161, 60, 50, 49\}$ $Mflops/s$ processors

For processor grids (1,4) and (1,5) a very good load balancing is achieved. For the other grids the three slower processors took approximately 15% less time than the other ones, due to the block indivisibility. The algorithm requires a significant number of communication points which results in a not very scalable algorithm as shown in figure 7 (left).

Fig. 7. Isogranularity curves for a 6 processor homogeneous machine connected by a 10 *Mbit/s* Ethernet; 160*K* elements for TRD, LU and QR and 250*K* for LU2

The scalability analysis was made in a homogeneous machine in order to reduce the influence of load unbalance.

5 Parallel Implementation of the Modal Matching Algorithm

This high level image processing algorithm [26] is applied for the tracking of deformable objects along a sequence of images. Figure 8 shows the application of the algorithm. It is based on finite element analysis and it requires the computation of the eigenvectors of symmetric matrices. The aim is to obtain correspondences between object points of image i and $i + n$. The algorithm is divided into eigenvector computation and matrix correlation. The eigenvector computation is subdivided into three operations: tridiagonalisation, correspondent orthogonal matrix and QR iteration. The parallelisation is then realised by the individual parallelisation of each operation. Data is redistributed if the processor grid changes between operations.

5.1 Tridiagonal Reduction and Orthogonal Matrix Computation

Tridiagonal reduction is the first algorithm applied to the symmetric matrix in order to obtain the eigenvectors. The algorithm output is a tridiagonal matrix T so that:

$$A = Q^T T Q \quad (4)$$

Fig. 8. Application of the modal analysis algorithm to a sequence of heart contours

The matrix T replaces A in memory. As shown in figure 9 the best grid is a row of processors. Details of the algorithm can be found in [8].

The matrix elements of T , apart from the tridiagonal positions, store the data required for the second step of the eigenvector algorithm, i.e. the computation of Q .

If the order of computation of the tridiagonal reduction was followed, an $O(n^4)$ algorithm would be obtained, corresponding to a matrix by matrix product in each step; $n - 2$ steps for a matrix of size (n, n) . However, the computation can be efficiently organised as described in [22] for a sequential algorithm, obtaining a scalable operation for the virtual machine. Figure 9 shows that the best grid is a row of processors.

5.2 Symmetric QR Iteration

The QR iteration is the last operation for the eigenvector computation. The aim is to obtain from the tridiagonal matrix T one diagonal matrix A where the elements are the eigenvalues of A :

$$T = G^T A G \quad (5)$$

The matrix G is then used to compute the eigenvectors Q' of A :

$$Q' = Q G^T \quad (6)$$

Matrix G^T is obtained by iterating and updating it with the Givens rotations [15]. To obtain Q' a matrix by matrix product would be required. However, the operations can be organised in order to update Q' in each iteration, avoiding the last matrix product. In the update only two columns of Q' are updated. Based on this fact a scalable operation was implemented by allowing the redistribution of data. The optimal data distribution is by blocks of rows so that any given row is completely allocated to a given processor, avoiding communications between processors for the update of Q' . The parallelisation implemented keeps the $O(n^2)$ chase operation in one processor which computes all rotations for an iteration, and distributes them over a column of processors. Then all processors update their rows, the $O(n^3)$ part, in parallel and without communications. This strategy has a huge impact in the scalability of the QR iteration as shown by

the isogranularity curve in figure 7. A good load balancing is also achieved for a heterogeneous machine as shown in figure 6.

The ideal grid for the QR iteration is the opposite (column vs. row) of the ones for tridiagonal and orthogonal matrix computation. This is the reason for considering indivisible operations and allowing redistribution of data between them to adapt the parallel machine to each operation.

5.3 Matrix Correlation

After the QR iteration has been computed for the objects in both images the eigenvectors are ordered in decreasing order of magnitude of the correspondent eigenvalue. The correlation operation measures the similarity between the eigenvectors of both objects. The behaviour of the processing time function shown in Figure 9 is different from the other operations. The best grid is either a row or a column of processors. The parallel algorithm is scalable as shown in figure 7.

Fig. 9. Estimated processing time for a 6 processor homogeneous machine connected by a 10 *Mbit/s* Ethernet

6 Results

Results are presented for machine M1 composed by 6 homogeneous processors of 141 *Mflop/s* each, $M2=\{244, 244, 161, 161, 60, 50, 49\}$ *Mflop/s* and $M3=\{161, 161, 112, 80\}$ *Mflop/s* processors. M1 is connected by a 10 *Mbit/s* Ethernet, and M2 and M3 by a 100 *Mbit/s* one. The performance metrics used to evaluate the parallel application are, first, the runtime, and second the speedup achieved. To have a fair comparison in terms of speedup, one defines the Equivalent Machine Number ($EMN(p)$) which considers the power available instead of the number of machines which, for a heterogeneous environment, is an ambiguous information. Equation 7 defines $EMN(p)$ and heterogeneous efficiency E_H , for p processors used, where S_1 is the computational capacity of the processor that executed the serial code, also called the master processor.

$$EMN(p) = \frac{\sum_{i=1}^p S_i}{S_1}, \quad E_H = \frac{Speedup}{EMN(p)} \quad (7)$$

For the machine M3 $EMN(4) = 3.19$, i.e. using 4 processors of the heterogeneous machine is equivalent to 3.19 processors identical to the master processor if this is the 161 *Mflop/s* one.

Fig. 10. Application results of the active contour algorithm

The right table of figure 10 presents results for the parallel active contour algorithm executed in the M3 machine for an image of 64 *KB* (figure 4) and for a 256 *KB* one (the left picture in figure 10). The time T_1 represents the processing time of the serial code in the master processor and T_{VM} the parallel processing time in the virtual machine. The number of processors selected in each step of the algorithm changes in order to minimise the processing time.

Fig. 11. Eigenvector computation in the M2 machine

Results for the eigenvector computation are presented in figure 11 for machine M2 due to the wide application of the algorithm. As shown, the heterogeneous

efficiency is near 80% for matrices with more than 1400^2 elements. However, the first metric is processing time which is reduced for matrices larger than 400^2 elements.

Fig. 12. Modal analysis in the homogeneous machine M1

To show the improvement due to the dynamic management of the parallel processing system, results for the modal analysis algorithm are presented for the homogeneous machine M1, figure 12. The left chart compares the computation time of the virtual machine VM when the optimal number of processors is selected, as indicated in the processing results table, against the processing time when the same number of processors are used for all stages of the algorithm. In the latter case the minimum time is obtained with 4 processors; however, the total time is higher than the time obtained with VM.

7 Conclusions

A operation based parallel image processing system for a cluster of personal computers was presented. The main objective is that the user of a computationally demanding application may benefit from the computational power distributed over the network, while keeping other active users undisturbed.

This goal can be achieved in a transparent manner for the user, once the modules of his/her application are correctly parallelised for the target network and the performance of the machines in the network is known. The application, before initiating a parallel module, determines the best available computer composition for a parallel virtual computer to execute it, and then launches the module, achieving the best response time possible in the actual network conditions.

Practical tests were conducted both on homogeneous and heterogeneous networks. In both cases the theoretically optimal computer grid was confirmed by the measured performance. A balanced load was achieved in both machines. The machine scalability depends essentially on the communication requirements of the operations. For QR iteration and matrix correlation the system is scalable; however, it is not so for the tridiagonal reduction.

Other generic modules will be parallelised and tested, so that an ever increasing number of image analysis methods may be assembled from them. Application domains other than image analysis may also benefit from the proposed methodology.

References

1. A. Alves, L. Silva, J. Carreira, and J. Silva. Wpvm: Parallel computing for the people. In *HPCN'95 High Performance Computing and Network Conference*. Springer-Verlag LNCS, <http://dsg.dei.uc.pt/wpvm>, 1995.
2. T. Anderson, D. Culler, D. Patterson, and The NOW Team. A case for now (network of workstations). *IEEE Micro*, February 1995.
3. D. Bader, J. JáJá, D. Harwood, and L. S. Davis. Parallel algorithms for image enhanced and segmentation by region growing with an experimental study. Technical Report UMCP-CSD:CS-TR-3449, University of Maryland, May 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. Canny. A computational approach to edge detection. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 8(6), November 1986.
7. 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.
8. 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.
9. 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.
10. J. W. Demmel. *Applied Numerical Linear Algebra*. SIAM, 1997.
11. H. Derin and C.-S. Won. A parallel image segmentation algorithm using relaxation with varying neighborhoods and its mapping to array processors. *Computer Vision, Graphics, and Image Processing*, (40):54–78, 1987.
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. J. Dongarra and D. Walker. The design of linear algebra libraries for high performance computers. Technical Report LAPACK Working Note 58, University of Tennessee, Knoxville, June 1993.
14. R. Geijn and J. Watts. Summa: Scalable universal matrix multiplication algorithm. Technical Report CS-95-286, University of Tennessee, Knoxville, 1995.
15. Gene Golub. *Matrix Computations*. The Johns Hopkins University Press, 1996.

16. J.F. JáJá and K.W. Ryu. The block distributed memory model. Technical Report CS-TR-3207, University of Maryland, January 1994.
17. Dan Judd, Nalini K. Ratha, Philip K. McKinley, John Weng, and Anil K. Jain. Parallel implementation of vision algorithms on workstation clusters. In *Proceedings of the International Conference on Pattern Recognition*, pages 317–321. IEEE Press, 1994.
18. Zoltan Juhasz and Danny Crookes. A pvm implementation of a portable parallel image processing library. In *EuroPVM 96*, volume 1156, pages 188–196. Springer-Verlag LNCS, 1996.
19. Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, pages 321–331, 1988.
20. Michael J. Litzkow, Miron Livny, and Matt W. Mutka. Condor - a hunter of idle workstations. In *Proc. IEEE 8th International Conference on Distributed Computing Systems*, pages 104–111. IEEE CS Press, 1988.
21. M. Luckenhaus and W. Eckstein. A thread concept for automatic task parallelization in image analysis. In *Proceedings of Parallel and Distributed Methods for Image Processing II*, volume 3452, pages 34–44. SPIE, 1998.
22. W.H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1997.
23. E. S. Quintana, G. Quintana, X. Sun, and R. Geijn. Efficient matrix inversion via gauss-jordan elimination and its parallelization. Technical Report CS-TR-98-19, University of Texas, Austin, September 1998.
24. L. P. Reis, J. Barbosa, and J. M. Sá. Active contours: Theory and applications. In *RECPAD96 - 8th Portuguese Conference on Pattern Recognition*, Guimarães, Portugal, 1996.
25. C. Seitz. Myrinet - a gigabit per second local-area network. *IEEE Micro*, February 1995.
26. L. Shapiro and J. M. Brady. Feature-based correspondence: an eigenvector approach. *Butterworth-Heinemann Lda*, 10(5), June 1992.
27. J. Shen and S. Castan. An optimal linear operator for step edge detection. *CVGIP: Graphical Models and Image Processing*, 54(2):112–133, March 1992.
28. H. J. Siegel, L. J. Siegel, F. C. Kemmerer, P. T. Mueller Jr., H. E. Smalley Jr., and S. D. Smith. Pasm: A partitionable simd/mimd system for image processing and pattern recognition. *IEEE Transactions on Computers*, C-30(12), December 1981.
29. S. M. Smith and J. M. Brady. Asset-2: Real-time motion segmentation and shape tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(8):814–820, 1995.
30. L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, August 1990.