Observations on Data Distribution and Scalability of Parallel and Distributed Image Processing Applications

ROMAN PFARRHOFER[1] AND ANDREAS UHL[1,2]
[1]School of Telematics & Network Engineering, CTI, Klagenfurt, AUSTRIA
[2]Department of Scientific Computing, Salzburg University, AUSTRIA
r.pfarrhofer@happy-snack.at, uhl@cosy.sbg.ac.at

Abstract: A MATLAB-based toolbox for efficient computing on Windows PC networks is employed for distributed template matching in an image database. We discuss the results from a data distribution and scalability viewpoint and derive interesting implications for parallel image processing applications on cluster systems.

Key-Words: parallel and distributed image processing, template matching, MATLAB

1 Introduction

In the recent years, computing with visual and multimedial data has emerged as a key technology in many areas. Applications include consumer electronics (Photo-CD, HDTV, SHDTV, Video-on-Demand, video games), medical imaging (e.g. digital radiography), video-conferencing, and scientific visualization.

However, the creation, processing, and management of these data types require an enormous computational effort, often too high for single processor architectures. Therefore, this fact taken together with the inherent data parallelism in visual data makes image and video processing natural application areas for parallel computing.

This fact is also reflected by a number of conferences and workshops exclusively devoted to these topics. The "Workshop on Parallel and Distributed Image Processing, Video Processing, and Multimedia (PDIVM)" [1] is an annual workshop co-organized by one of the authors in the framework of the "International Parallel and Distributed Processing Symposium (IPDPS)". "Parallel and Distributed Methods for Image Processing I – IV" is an annual conference organized in the context of SPIE's annual meeting (published so far as SPIE proceedings no. 3166, 3452, 3817, and 4118). Also, several special sessions or topics at various conferences have been devoted to these or similar topics. For example, the "EuroPar" Conference series features one or two streams devoted to multimedia techniques each year. Finally, a special issue of the "Parallel Computing" journal on "Parallel Computing in Image and Video Processing" has been published recently (vol. 28(7-8), 2002).

Software based approaches are becoming more popular in this area (beside the use of DSP chips, FPGAs, media processors, or application specific VLSI) because of the performance increase of general-purpose processors and the rapid evolution of multimedia techniques which has dramatically shortened the time available to come up with a new hardware design for each improved standard or technique.

With the emerge of cluster computing and the corresponding potential availability of HPC systems in many universities and companies, this architecture is a cheap and highly available and yet powerful architecture for image processing applications [4]. However, applications requiring a large amount of synchronization and data transfer (like e.g. current tree based wavelet image coding schemes) can not be implemented efficiently on such systems [2]. Additionally, the explicit programming style required by message passing libraries for clusters (like PVM, MPI, or any other suited software system) makes program developement difficult and therefore this remains mostly a task for specialists. The moderate parallel multiprocessor architecture (i.e. shared memory MIMD) – often also denoted SMP – is an interesting (but much more expensive) alternative to multicomputers for this type of image processing tasks due to the availability of physically shared data space and more comfortable programming environments for parallel processing on such architectures (e.g. OpenMP, JAVA Threads) [9].

MATLAB has established itself as the numerical computing environment of choice on uniprocessors for a large number of engineers and scientists. In the context of image processing the most important reason is that MATLAB provides users with easy access to an extensive library of high quality numerical routines which can be used in a dynamical way and may be easily extended and integrated into existing applications.

In this work we discuss the use of the MDICE MATLAB toolbox for template matching in an image database on a cluster of workstations. In particular, we focus onto data distribution and scalability issues which arise in the context of load balancing. In section 2, we discuss the principles of MDICE. Section 3 covers results of distributed template matching and discusses the interesting results. Section 4 concludes the paper.

[1] `http://www.cosy.sbg.ac.at/~uhl/pdivm.html`

2 MATLAB Cluster Computing: MDICE

For most image processing applications, the desired levels of performance are only obtainable on parallel or distributed computing platforms. With the emerge of cluster computing, the demand for a solution to employ MATLAB on such systems is obvious. A comprehensive and up-to-date overview on high performance MATLAB systems is given by the "Parallel MATLAB Survey"[2]. Several systems may be downloaded from the Matworks server. There are basically three distinct ways to use MATLAB on HPC architectures:

1. Developing a high performance interpreter
   (a) Message passing: communication routines usually based on MPI or PVM are provided. These systems normally require users to add parallel instructions to MATLAB code [5].
   (b) "Embarrassingly parallel": routines to split up work among multiple MATLAB sessions are provided in order to support coarse grained parallelization. Note that the PARMATLAB and TCPIP toolboxes our own development is based upon fall under this category.
2. Calling high performance numerical libraries: parallelizing libraries like e.g. SCALAPACK are called by the MATLAB code [8]. Note that parallelism is restricted within the library and higher level parallelism present at algorithm level cannot be exploited with this approach.
3. Compiling MATLAB to another language (e.g. C, HPF) which executes on HPC systems: the idea is to compile MATLAB scripts to native parallel code [1, 7]. This approach often suffers from complex type/shape analysis issues.

Note that using a high performance interpreter usually requires multiple MATLAB clients whereas the use of numerical libraries only requires one MATLAB client. The compiling approach often does not require even a single MATLAB client. On the other hand, the use of numerical libraries and compiling to native parallel code is often restricted to dedicated parallel architectures like multicomputers or multiprocessors, whereas high performance interpreters can be easily used in any kind of HPC environment. This situation also motivated the development of our custom high performance MATLAB environment: since our target HPC systems are (heterogenous) PC clusters running a Windows system based on the NT architecture, we are restricted to the high performance interpreter approach. However, running a MATLAB client on each PC is expensive in terms of licensing fees and computational resources. MDICE [6] requires one MATLAB client (on the server machine) for distributed execution only.

MATLAB-based **DI**stributed **C**omputing **E**nvironment (MDICE) is based on the PARMATLAB and TCPIP toolboxes. The PARMATLAB toolbox supports coarse grained parallelization and distributes processes among MATLAB clients over the intranet/internet. Note that each of these clients must be running a MATLAB daemon to be accessed. The communication within PARMATLAB is performed via the TCPIP toolbox. Both toolboxes may be accessed at the Mathworks ftp-server.

However, in order to meet the goal to get along with a single MATLAB client the PARMATLAB toolbox needed to be significantly revised. The main idea was to change the client in a way that it can be compiled to a standalone application. At the server, jobs are created and the solve routine is compiled to a program library (*.dll). The program library and the datasets for the job are sent to the client. The client is running as background service on a computer with low priority. For this reason the involved client machines may be used as desktop machines by other users during the computation (however, this causes the need for a dynamic load balancing approach of course). This client calls over a predefined standard routine the program library with the variables sent by the server and sends the output of the routine back to the server. After the receipt of all solutions the server defragments them to an overall solution. The client-server approach is visualized in Fig. 1.

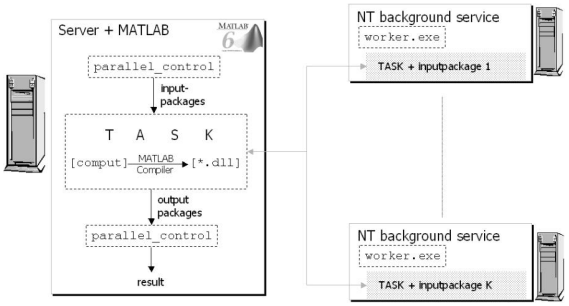

Figure 1: Client-server concept of MDICE.

The communication functionalities of the PARMATLAB toolbox have been extended as well. For example, in case of fault-prone file transmission (e.g. no space left on the clients' hard disk) the server is immediately notified about the failure. The underlying TCPIP toolbox requires all data subject to transmission to be converted into strings. For large amounts of data this is fairly inefficient in terms of memory demand and computational effort. In this case, we store the data as MAT-file, compress it (since these files are organized rather inefficiently), and finally convert it

[2] http://supertech.lcs.mit.edu/~cly/survey.html

into strings. After the computation is finished and the result has been sent, a new job may be processed by the client. Note that the *.dll library and constant variables do not have to be resent since the client informs the server about its status.

MDICE does not support any means of automatic parallelization or automatic data distribution. The user has to specifiy how the computations and the associated data have to be distributed among the clients. The same is true of course for the underlying PARMATLAB toolbox. Note that using MDICE on a PC cluster system for distributed image processing is an extreme case in terms of high communication cost.

3 Distributed Search in Image Databases

3.1 Template Matching

An image database is searched for a specific template, e.g. in our case the eye of Lena in the corresponding image (see Fig. 2). We consider images of 256×256 and 512×512 pixels, the size of the corresponding templates is 16×16 and 32×32 pixels, respectively.

Figure 2: Lena image with template.

We add noise to the selected region in the image and rotate it a random amount of degrees. Subsequently, the server creates 90 versions of the template, each rotated 90 degrees, and sends these templates to the clients where the remaining 270 rotated versions are created locally using the MATLAB command `rot90`. To facilitate distributed search, the image database is partitioned into a user-selected number of jobs and the jobs (i.e. images or image tiles) are distributed to be processed independently on the clients. If the number of jobs exceeds the number of images, these need to be tiled accordingly. Currently, only an integer multiple of the number of images is supported for the number of jobs. Note that in order to guarantee correct results the images need to be distributed in a redundant way (see Fig. 3 for a tiling) where the size of the overlap has to be the width of the template-1 in each dimension.

Figure 3: Image tiling example.

Note that the alternative would be to exchange the required border data among the clients. This approach is called "data swapping" for example in the case of parallel wavelet transforms [2]. Given the high communication cost on our target system overlapped tilting is much more efficient of course.

It is important to notice that in case of q tiles per image where q is a prime the images are tiled along one dimension only. According to [4] this approach is called "row partitioning". Note that row partitioning leads to more overlapping data as compared to the common case where both dimensions are tiled. In case q is non-prime, we use the factorization of q where the magnitude of the two dimensions of each (equally sized) tile should be as close as possible (see Fig. 3. This simple strategy overcomes the limitations concerning the admissible job number as imposed by the "cross partition" approach [4]. It should be pointed out that the "heuristic partitioning" approach [4] avoids tiling along one dimension only in case of q is prime. However, this comes at a high computational cost to compute the partitioning strategy and an irregular mapping pattern.

At the client, for each of the possible positions of all 360 possible rotated versions of the template in the search area we compute a normalized cross-

correlation [3, pp. 316-317]

$$c(u,v) = \sum_{x,y} f(x,y)t(x-u, y-v)$$

where $f(x,y)$ denotes the image, t is the template at position (u,v), and the sum is computed over x and y covering the range of the template. The rotation with the highest value for $c(u,v)$ is recorded and its position and magnitude is send to the server. At the end of the computation the server selects the area with the highest value of all jobs as the targeted image region.

3.2 Experimental Settings

The computational tasks of the applications subject to distributed processing are split into a certain number of equally sized jobs N to be distributed by the server among the M clients (usually $N \geq M$). Whenever a client has sent back its result to the server after the initial distribution of M jobs to M clients, the server assigns another job to this idle client until all N jobs are computed. This approach is denoted "asynchronous single task pool method" and facilitates dynamic load balancing in case of $N \gg M$. The computing infrastructure consists of the server machine (1.99 GHz Intel Pentium 4, 504 MB RAM) and the client machines (996 MHz Intel Pentium 3, 128 MB RAM), both types under Windows XP Prof. The Network is 100 MBit/s Ethernet. Note that the sequential reference execution times used to compute speedup have been achieved on a 996 MHz client machine with a compiled (not interpreted) version of the application to allow a fair comparison since the client code is compiled as well in the distributed application. We use MATLAB 6.5.0 with the MATLAB compiler 3.0 and the LCC C compiler 2.4.

3.3 Experimental Results

Fig. 4 shows the speedup of this application when varying the problemsize (# of images in the database) and keeping the number of jobs distributed among the clients fixed (16). We notice plateaus resulting from load distribution problems (e.g., 16 jobs may not be efficiently distributed among 14 clients whereas this is obviously possible for 16 clients). Lower speedup and less pronounced plateaus are exhibited in case of smaller problem size. Here, the expensive initial communication phase (where the server has to send the compiled code and input data to each of the clients) covers a significant percentage of the overall execution time. This leads to a significantly staggered start of the computation phases at different clients which dominates the other load inbalance problems.

Increasing the problem size leads to an increased speedup, as it is expected. However, the reason can not be the improved computation/communication ratio. Although the number of messages sent is constant when keeping the number of jobs fixed, the amount of data sent within each message is increased by the factor the database is enlarged. The reason for the increased speedup is the decreasing amount of overlapping data required for generating the correct redundant tiling. For example, when processing 2 images each image is partitioned into 8 tiles whereas in case of 8 images each image is cut into two halves only. Clearly, the amount of overlapping data is a factor 4 larger in the first case. It is important to notice that a higher amount of overlapping data does not only imply a higher communication amount but causes also (and more importantly) a higher amount of computation. As a consequence, increasing the number of jobs to get rid of the speedup plateaus or to achieve balanced load as in case of heterogeneous environments is obviously not a sound approach in this context. This is confirmed by the values for 8 images and 48 jobs in Fig. 4. Employing 14 clients a much better performance would be expected for 48 jobs as compared to 16 jobs due to the significantly better balanced load, however, this virtual advantage is destroyed by the higher amount of overlapping data and the associated computations.

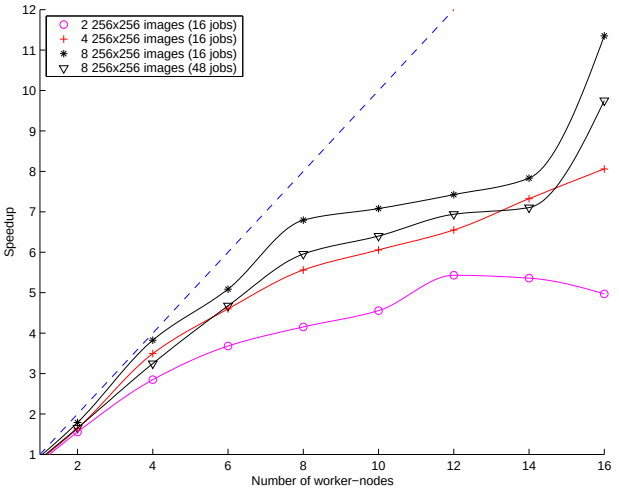

Figure 4: Results of template matching: speedup with varying problem-size.

Fig. 5 shows the time demand of distributed execution when varying the number of jobs (in integer multiples of the number of images involved) while keeping the amount of work and the number of clients fixed (i.e. a database consisting of eight 512×512 pixels images and 10 clients). The sequential execution time is 8740 seconds. Interestingly, we do not see improved performance for a larger number of jobs. The execution time using 8 jobs only (where 2 clients do not contribute to the computation at all) is the best value. The time behaviour for increasing the number of jobs seems to be highly irregular and hardly predictable.

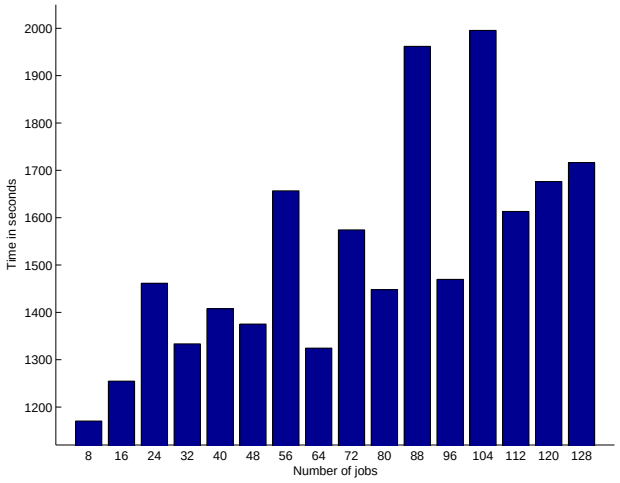

Figure 5: Results of template matching: varying the number of jobs.

On the one hand, increasing the number of jobs should help to improve the load distribution during the execution. This effect is shown in Fig. 6 where we display corresponding results of a Monte Carlo application using MDICE in the same environment [6]. For the test configuration considered, the optimal number of jobs is identified to be around 60. A further increase leads to an increase of execution time (caused by communication overhead) as well as a lower number causes worse results (caused by load inbalance).

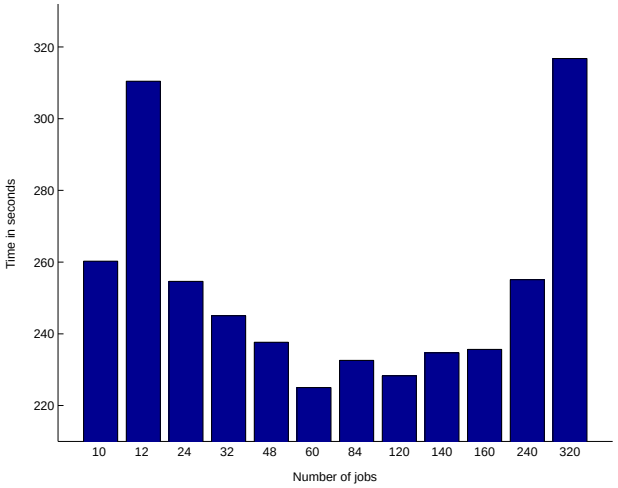

Figure 6: Results of Monte Carlo simulation with varying job number.

On the other hand, in case of image tiling, a large number of jobs degrades the computation/communication ratio (more messages, same computation) and causes a large amount of overlapping data in the tiling (which causes larger messages and a higher amount of overall computations). However, this tradeoff does not explain the irregular results in Fig. 5. There is one more important aspect to be considered. The number of jobs devided by the number of images in the database gives the number of tiles required per image. As already previously mentioned, if the resulting number is a prime, the image is tiled along one dimension only which results in a higher amount of overlapping data as in case of tiling along both dimensions. The resulting higher amount of computation in the prime case can be clearly seen in Fig. 5: poor performance is especially exhibited for 24, 40, 56, 88, and 104 jobs (which corresponds to 3, 5, 7, 11, and 13 tiles per image, respectively). Fig. 7 illustrates this fact by showing the amount of additional work in % caused by image tiling. The larger values in the prime case are clearly shown.

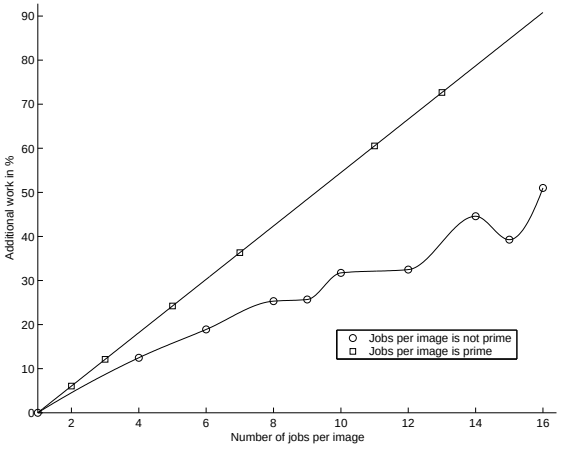

Figure 7: Comparison of additional workload caused by tiling using prime and non-prime job numbers.

Furthermore, the behaviour within the two classes "prime number of tiles" and "non-prime number of tiles" is not entirely regular as well. Figs. 8 and 9 show a visualization of the distributed execution, where black areas represent computation time-intervals, gray areas communication events, and white areas idle times. The x-axis shows the time (in seconds), y-axis is covered by the different clients. Eight 512×512 pixels images are processed.

For example, one would expect better behaviour for 24 jobs (Fig. 8) as compared to 40 jobs (Fig. 9) due to the lower amount of overlapping data. However, the 40 job setting is faster. Obviously, the higher amount of computation and communication caused by 40 jobs is more than compensated by the perfect distribution of 40 jobs among 10 clients.

4 Conclusion

Using distributed template matching as a sample application, we have observed the following facts in the context of data parallel image processing involving neighbourhood operations on a cluster architecture:

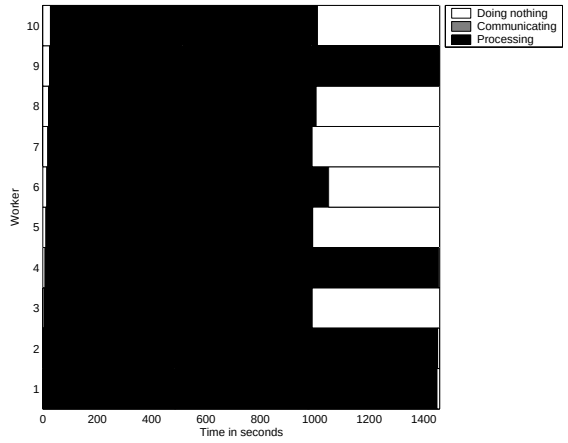

Figure 8: Visualization of load distribution with 24 jobs on 10 clients.

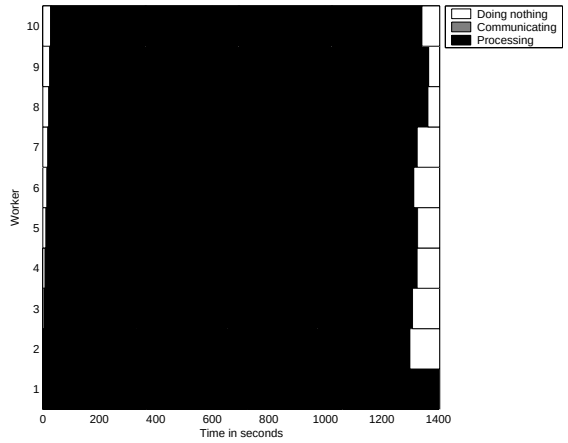

Figure 9: Visualization of load distribution with 40 jobs on 10 clients.

1. Increasing the number of jobs (i.e. image tiles to be processed) does not only increase the communication amount but also the computational effort is increased since the amount of overlapping data grows. This has two major implications:
   (a) Increasing the number of jobs to achieve better balanced load might be contraproductive and result in worse performance.
   (b) Increasing the number of processing elements when keeping the problem size fixed does not lead to better performance at some point (poor scalability).
2. The question how the image data is partitioned is crucial with respect to performance in case the application involves neighbourhood operations. Simple partitioning along one dimension only (row partitioning) is shown to perform significantly worse as compared to tiling both image dimensions.

Acknowledgements

This work has been partially supported by the Austrian Science Fund FWF, project no. 13907.

References

[1] L. DeRose and D. Padua. A MATLAB to Fortran 90 translator and its effectiveness. In *Proceedings of 10th ACM International Conference on Supercomputing*. ACM SIGARCH and IEEE Computer Society, 1996.

[2] M. Feil and A. Uhl. Wavelet packet zerotree image coding on multicomputers. In *Proceedings of the 10th Euromicro Workshop on Parallel and Distributed Processing*, pages 353–359. IEEE Computer Society, 2002.

[3] R.M. Harmlick and L.G. Shapiro. *Computer and Robot Vision, Vol. III*. Addison-Wesley, 1992.

[4] C. Lee and M. Hamdi. Parallel image processing applications on a network of workstations. *Parallel Computing*, 21:137–160, 1995.

[5] S. Pawletta, T. Pawletta, and W. Drewelow. Comparison of parallel simulation techniques - MATLAB/PSI. *Simulation News Europe*, 13:38–39, 1995.

[6] R. Pfarrhofer, P. Bachhiesl, M. Kelz, H. Stögner, and A. Uhl. MDICE - a MATLAB toolbox for efficient cluster computing. In G.R. Joubert, W.E. Nagel, F.J. Peters, and W.V. Walter, editors, *Parallel Computing: Software Technology, Algorithms, Architectures, and Applications. Proceeding of the 10^{th} ParCo Conference in Dresden, 2003*. Elsevier B.V., 2003. To appear.

[7] M. Quinn, A. Malishevsky, N. Seelam, and Y. Zhao. Preliminary results from a parallel MATLAB compiler. In *Proceedings of the International Parallel Processing Symposium (IPPS)*, pages 81–87. IEEE Computer Society Press, 1998.

[8] S. Ramaswamy, E.W. Hodges, and P. Banerjee. Compiling MATLAB programs to SCALAPACK: Exploiting task and data parallelism. In *Proceedings of the International Parallel Processing Symposium (IPPS)*, pages 814–821. IEEE Computer Society Press, 1996.

[9] C. Rothlübbers and R. Orglmeister. Parallel image processing using a Pentium based shared-memory multiprocessor system. In H. Shi and P.C. Coffield, editors, *Parallel and Distributed Methods for Image Processing*, volume 3166 of *SPIE Proceedings*, pages 46–54, 1997.