Towards User Transparent Data and Task Parallel Image and Video Processing: An Overview of the Parallel-Horus Project

F.J. Seinstra, D. Koelma, and A.D. Bagdanov

Intelligent Sensory Information Systems,
Faculty of Science, University of Amsterdam,
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
(fjseins, koelma, andrew)@science.uva.nl

Abstract. In the research area of image and video processing, the *Horus library* [5] has become a valuable system for software driven rapid prototyping, and an essential vehicle for knowledge transfer at the level of universities and expertise centers. Due to its strictly sequential implementation, however, Horus can not always satisfy all demands for high performance. As computational requirements for the processing of large image sets and video streams continue to increase, it is essential to provide a Horus implementation that applies to high performance computers.
As researchers in image and video processing can not be expected to also become experts in parallel computing, the *Parallel-Horus* project aims to shield Horus users from *all* intrinsic complexities of parallelization. This paper presents an overview of the Parallel-Horus project; it discusses the project's goals, as well as current and future research directions. Also, the efficiency of the current Parallel-Horus implementation is demonstrated by evaluating a strictly sequential state-of-the-art imaging application.

1 Introduction

The complexity and scale of scientific problems in many different research areas has been a driving force for the development of ever faster computer architectures. To effectively exploit the available processing power of such systems, it is essential to have a thorough understanding of their architectural complexity. Despite this requirement, large numbers of non-expert users are still tempted by the potential processing speeds — generally to emerge with nothing but a disappointing result. In [3] this problem is stated somewhat more dramatically as follows:

> Anecdotal reports abound about researchers with scientific and engineering problems who have tried to make use of parallel processing systems, and who have been almost fatally frustrated in the attempt.

Clearly, there is a major discrepancy between the desire to easily obtain increased performance, and the potential of high performance systems to satisfy this desire.

The specific research area of image and video processing — which is the field of focus of this paper — also demonstrates a persistent desire to access the speed potential of high performance computers. The desire partially stems from the fact that it has been recognized for years that the application of parallelism in imaging can be highly beneficial [21]. Yet, in image processing research the discrepancy between desire and reality is no less severe.

We ascribe this problem primarily to the fact that no programming tool is available that can effectively help non-expert parallel programmers in the development of efficient high performance imaging applications. Existing programming tools generally require the user to identify the available parallelism at a level of detail that is beyond the skills of non-expert parallel programmers [11, 14, 16]. Thus, the key to fast algorithm design and evaluation on large data sets, as well as to rapid high performance image and video application development, lies in the availability of a familiar programming tool that shields the image processing researcher from all intrinsic difficulties of parallel implementation.

Our work (referred to as the *Parallel-Horus* project, see also [13–16]) is an attempt to effectively bridge the gap between the expertise of the image processing community, and the additional expertise required for efficient employment of high performance computers. More specifically, the project aims at designing and implementing a software architecture that allows non-expert parallel programmers to develop image and video processing applications for efficient execution on Beowulf-type commodity clusters [19]. Hence, the Parallel-Horus project addresses the following fundamental research issue: *how to design a sustainable, yet efficient software architecture for data and task parallel image and video processing, that provides the user with a fully sequential programming model, and hides all parallelization and optimization issues from the user completely.*

This paper presents an overview of the Parallel-Horus project. Apart from presenting the project's aims, it discusses directions for current and future research. In addition, the efficiency of the current Parallel-Horus implementation is demonstrated by evaluating a strictly sequential state-of-the-art imaging application.

2 Parallel-Horus: Design for Sustainability and Efficiency

In the literature, a trend in the design of parallelization aids for high performance image and video processing is to provide a *software library* consisting of a set of pre-parallelized routines, and hide as much as possible the complexities of parallelization behind the library's API. Due to the relative ease of implementation, a multitude of such libraries exists [3, 4, 6–8, 18, 20]. An important design goal in much of this research is to provide operations that have optimal efficiency on a range of parallel machines. In general, this is achieved by hard-coding a number of different parallel implementations for each operation, one for each platform. Unfortunately, the creation of a parallel library in this manner has several major drawbacks. First, manually creating multiple parallel versions of the many operations commonly applied in image processing research is a laborious task. Second, obeying to requests for library extensions becomes even more troublesome than in the sequential case. Third, as new target platforms are made available at regular intervals, code maintenance becomes hard — if not impossible — on the long term. Finally, with each library expansion it becomes ever more difficult to incorporate a single elegant performance optimization strategy for full applications. For these reasons we take a different approach.

In the development of the Parallel-Horus architecture we strive to enhance sustainability by minimizing the implementation effort, without compromising on the efficiency of execution. For one, this is achieved by recognizing that there is a limited number of ways in which the pixels in images and video streams can be processed to produce meaningful results. Important in this respect is the classification of imaging operations made in Image Algebra [12]. Originating from this classification, a sequential image and video processing library is currently being implemented at the University of Amsterdam: *Horus* [5]. The Horus library is highly innovative in its design and implementation, most significantly because the library's core comprises of a very small set of so-called *algorithmic patterns*. The primary importance of these patterns is that each serves as a template operation for a large set of image operations with similar behavior, which greatly reduces the implementation effort. Also, the algorithmic patterns abstract from the actual datatype each operation is applied upon, to avoid a combinatorial explosion of code that deals with all possible kinds of image datatypes.

The next important step is to recognize that, for parallel implementation of each algorithmic pattern, much of the related sequential code can be reused. To that end, for each sequential algorithmic Horus pattern we have defined a so-called *parallelizable pattern* [16]. Such pattern constitutes the maximum amount of code of an algorithmic pattern that can be performed both sequentially and in parallel — in the latter case without having to communicate to obtain non-local data. Also, further reduction of the software engineering effort is achieved by implementing the parallelizable patterns such that they are capable of adapting to the performance characteristics of the parallel machine at hand [13, 16].

The final step in reaching our goal is to recognize that for high performance of full applications it is not sufficient to consider parallelization of library operations *in isolation*. As shown in [15], it is possible to extend any library implemented on the basis of parallelizable patterns with a very efficient finite state machine based run-time system for communication minimization of applications running in data parallel fashion. Moreover, the run-time system that was integrated in the Horus library according to this approach proved to deliver close-to-optimal parallel performance for many strictly sequential imaging applications [14].

3 Parallel-Horus: Large-scale Image & Video Processing

In the early days of image processing research, algorithms and applications had modest computational requirements. In general not more than a few small-sized low-dimensional images needed to be processed. In contrast, today's research in image and video processing is often confronted with large collections of large-sized and high-dimensional images, and hours of high resolution video. In addition, thorough evaluation (thus: calculation) of large algorithmic parameter spaces on image data sets is now commonplace.

Partially, this shift in computational requirements is driven by advances in techniques for high resolution image acquisition, e.g. in biomedical imaging [23] and light microscopy [10]. Also, applications requiring interactive access to large image collections are becoming ever more abundant [17, 22]. Finally, in the presentation of results obtained from imaging experiments there is a trend to incorporate evaluations on larger data sets. This is due to the fact that many imaging algorithms perform well on a limited set of (artificial) images, but are much less applicable to other (often real-world) images. Thus, to investigate the limitations and potential of algorithms, large-scale evaluations are essential. To this end, *the main focus of the Parallel-Horus project is on high performance processing of large image data sets, video streams, and large algorithmic parameter spaces.*

On the one hand, support for this type of processing is delivered in the form of pre-parallelized library operations, which are integrated in the existing sequential Horus library (component ① in Figure 1). Also, the run-time system for communication minimization (as referred to above) is integrated in the Horus library and to be expanded to cover all functionality for the processing of large parameter spaces, data sets, and video streams (component ② in Figure 1).

On the other hand, the usage of a single library operation to initiate the processing on a set of images, or a complete parameter space, may not fit the image processing researcher's programming and specification preferences. Instead, it is expected that many developers of imaging applications will keep to the more familiar specification of a chain of events by way of loop constructs provided by the host programming language (i.e. C or C++) [11]. Such constructs often incorporate inherent potential for (independent) parallel processing of each loop

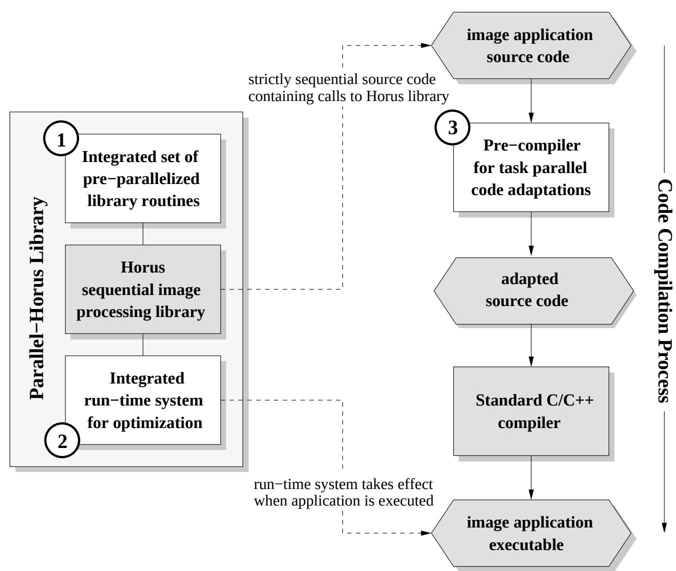


Fig. 1. *Simplified overview of the Parallel-Horus architecture.*

iteration. As this type of parallelism can not be included in the image library itself, initial solutions are to be incorporated in the form of macros or compiler directives for parallelization, similar to those described by Oliveira et. al. [9]. However, as the imaging researcher should not personally have to include such parallelization hints in a sequential application, at a later stage in the project code adaptations are to be generated automatically by means of a pre-compiler (see component ③ in Figure 1). The pre-compiler is to analyze the strictly sequential image processing code, and insert task parallelization macros wherever needed, to alleviate the programmer from personally having to perform this task.

4 Parallel-Horus: Towards Mixed Data/Task Parallelism

Parallel-Horus extends our earlier research, which was aimed at the design of a software architecture for user transparent data parallel processing of single (or just a few) images on Beowulf clusters. *A major aim of the Parallel-Horus project is to integrate task parallelism within the same software architecture to facilitate high performance processing of a much larger class of applications.*

Our earlier restriction to data parallelism was sensible for many reasons. First, the application of data parallelism is a natural approach for image processing, as many operations require the same function to be applied to each individual element of an image data structure. Second, in applications that need to process just few images, the number of independent tasks is generally small, especially when compared to the fairly abundant number of elements present in the image structures to be processed. Another important reason is that load balancing is generally much more difficult in the task parallel model. The decisive factor for not having integrated task parallelism in our work so far, however, was the expected difficulty of combining the two programming paradigms in a fully user transparent parallel environment.

The situation has changed, however. Because our previous work has shown that it is possible to design a fully user transparent architecture for data parallel image processing with relative ease [13–16], the task of integrating the two parallel paradigms into a single framework suddenly becomes feasible. The reason for this is as follows. In the current Parallel-Horus system any part of a strictly sequential program runs in data parallel mode, whenever the available number of processors is known. In other words: for any sequential program implemented using our architecture, data parallelism comes for free. In its simplest form, combined data and task parallelism can now be achieved easily by designating a certain number of nodes to each loop iteration (i.e. data parallelism), and let these iterations run concurrently (i.e. task parallelism). Preliminary work in this direction shows that very simple macro definitions are sufficient to let independent loop iterations indeed be executed independently, and even provide partial load balancing without extensive code analysis. Although the resulting mixture of data and task parallelism is (as of yet) restricted, the ease with which these results are obtained is remarkable. Therefore we strongly believe that further investigations in this direction are not only important, but also will be rewarding.

5 An Example Application

To evaluate the current Parallel-Horus implementation, this section describes the (data) parallel execution of an example imaging application: i.e. the extraction of rectangular size distributions from document images. Prototypical code of the Parallel-Horus architecture, as wel as for this and many other example applications, is available at `http://www.science.uva.nl/~fjseins/ParHorusCode/`.

Measurements were performed on the 72-node Distributed ASCI Supercomputer 2 located at the Vrije Universiteit in Amsterdam [2]. All nodes consist of two 1-Ghz Pentium-III CPUs, with 2 GByte of RAM, and are connected by a Myrinet-2000 network; each node ran the RedHat Linux 7.2 operating system. Our software architecture was compiled using `gcc` 2.96 and linked with MPICH-GM, which uses Myricom's GM as its message passing layer on Myrinet.

Rectangular Size Distributions

As discussed in [1], *rectangular size distributions* are an effective way to characterize visual similarities between document images. A particularly efficient sequential implementation for this problem, using recursive filters, is presented in Listing 1. It should be noted that such recursive filter operations make for a sequential implementation which is notoriously hard to parallelize efficiently.

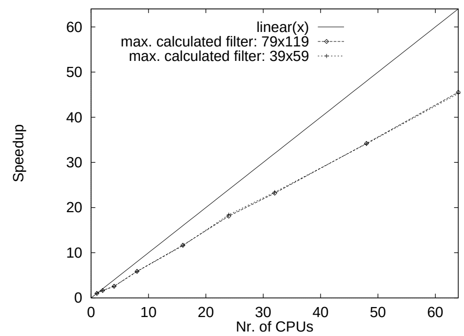
Due to the Parallel-Horus architecture, the sequential code of Listing 1 directly constitutes a parallel program as well. Measurement results for the gener-

```
calculateRectangularSizeDistribution(IMAGE inIm, INT w, INT h) {
  vertIm = verDist(inIm, 0);
  area = reduceOp(inIm, "sum");
  FOR (y=0; y≤h; y++) DO
    oy = (y/2h)*(inIm.height+1);
    vThreshIm = horDist(binPixOpC(vertIm, oy, "grtrthan"), 0);
    filtered = -1;
    FOR (x=0; x≤w; x++) DO
      IF (filtered != 1.0) THEN
        ox = (x/2w)*(inIm.width+1);
        hThreshIm = binPixOpC(vThreshIm, ox, "lessequal");
        hThreshIm = binPixOpC(verDist(hThreshIm, MAXVAL), oy, "grtrthan");
        hThreshIm = binPixOpC(horDist(hThreshIm, MAXVAL), ox, "lessequal");
        filtered = (area - reduceOp(hThreshIm, "sum")) / area;
      FI
      ... and save 'filtered' for current x,y combination ...
    OD
  OD
}
```

Listing 1: *Pseudo code for fast calculation of rectangular size distributions; maximum size of calculated filters denoted by 'w' and 'h'. Functions 'horDist' and 'verDist' perform horizontal and vertical distance transforms, using recursive filter-pairs [1].*

# CPUs	'39x59' (s)	'79x119' (s)
1	1263.277	4589.818
2	779.486	2821.391
4	497.145	1789.319
8	216.679	781.175
16	109.284	393.763
24	68.652	253.647
32	53.967	197.962
48	37.048	134.180
64	27.926	100.792

(a)



(b)

Fig. 2. *(a) Performance and (b) speedup for computing rectangular size distributions for document image of size 2325×3075 (2-byte) pixels. Maximum size of calculated filters either 39×59 or 79×119. Speedup lines for either measurement essentially coincide.*

ated parallel version of the presented program are given in Figure 2. The results show that, for realistic images of size 2325×3075 pixels, our architecture provides very good speedup characteristics: 45.5 on 64 processors — an efficiency of 71.2%. Therefore, we conclude that the Parallel-Horus architecture provides high parallel efficiency for the presented image processing problem. Remember, for the application programmer these results come without any parallelization effort whatsoever. More results for other imaging applications are found in [14, 16]

6 Conclusions

The Parallel-Horus project aims at the development of an effective programming tool that provides sustainable support in the implementation of data and task parallel image and video processing software by non-experts in high performance computing. With the implementation of the current Parallel-Horus architecture, which supports fully user transparent data parallel execution, we have largely succeeded in that mission. However, strictly data parallel execution does not always deliver sufficiently high performance. Also, due to the need for independent processing of large amounts of data, many state-of-the-art multimedia applications have a much higher degree of inherent task parallelism than before. Therefore, we are currently working on extensions that allow for integration of efficient task parallelism within the same Parallel-Horus framework. Solutions are to be provided in the form of library operations for the processing of large data sets, as well as through a pre-compiler that is to make straightforward task parallel adaptations to strictly sequential image processing source codes. In the end, the availability of a combined data and task parallel environment for user transparent high performance image and video processing will certainly have a stimulating effect on the investigation of the many computationally demanding problems the image processing community is so often confronted with.

References

1. A.D. Bagdanov et al. Multi-scale Document Description using Rectangular Granulometries. In *Document Analysis Systems V*, pages 445–456, August 2002.
2. H.E. Bal et al. The Distributed ASCI Supercomputer Project. *Operating Systems Review*, 34(4):76–96, October 2000.
3. L.H. Jamieson et al. A Software Environment for Parallel Computer Vision. *IEEE Computer*, 25(2):73–75, February 1992.
4. Z. Juhasz et al. A PVM Implementation of a Portable Parallel Image Processing Library. In *Proc. EuroPVM'96*, pages 188–196, Munich, Germany, October 1996.
5. D. Koelma et al. Horus C++ Reference, Version 1.1. Technical report, ISIS, Faculty of Science, University of Amsterdam, The Netherlands, January 2002.
6. C. Lee and M. Hamdi. Parallel Image Processing Applications on a Network of Workstations. *Parallel Computing*, 21(1):137–160, January 1995.
7. P.J. Morrow et al. Efficient Implementation of a Portable Parallel Programming Model for Image Processing. *Concurrency: Pract. & Exp.*, 11:671–685, 1999.
8. C. Nicolescu and P. Jonker. A Data and Task Parallel Image Processing Environment. *Parallel Computing*, 28(7–8):945–965, August 2002.
9. P. Oliveira and H. du Buf. SPMD Image Processing on Beowulf Clusters: Directives and Libraries. In *Proc. IPDPS 2003*, Nice, France, April 2003.
10. P. van Osta et al. The Principles of Scale Space Applied to Structure and Colour in Light Microscopy. *Proc. Royal Microscop. Soc.*, 37(3):161–166, September 2002.
11. C.M. Pancake and D. Bergmark. Do Parallel Languages Respond to the Needs of Scientific Programmers? *IEEE Computer*, 23(12):13–23, December 1990.
12. G.X. Ritter and J.N. Wilson. *Handbook of Computer Vision Algorithms in Image Algebra*. CRC Press, Inc, 1996.
13. F.J. Seinstra and D. Koelma. P-3PC: A Point-to-Point Communication Model for Automatic and Optimal Decomposition of Regular Domain Problems. *IEEE Transactions on Parallel and Distributed Systems*, 13(7):758–768, July 2002.
14. F.J. Seinstra and D. Koelma. User Transparency: A Fully Sequential Programming Model for Efficient Data Parallel Image Processing. *Concurrency and Computation: Practice and Experience*, 16(6):611–644, May 2004.
15. F.J. Seinstra, D. Koelma, and A.D. Bagdanov. Finite State Machine Based Optimization of Data Parallel Regular Domain Problems Applied in Low Level Image Processing. *IEEE Transactions on Parallel and Distributed Systems (in press)*, 2004.
16. F.J. Seinstra, D. Koelma, and J.M. Geusebroek. A Software Architecture for User Transparent Parallel Image Processing. *Parallel Computing*, 28(7–8):967–993, August 2002.
17. C.G.M. Snoek and M. Worring. Multimodal Video Indexing: A Review of the State-of-the-art. *Multimedia Tools and Applications*, 2004. (in press).
18. J.M. Squyres et al. Cluster-Based Parallel Image Processing. Technical report, Lab. for Scientific Computing, Univ. Notre Dame, Indiana, USA (TR 96-9), 1996.
19. T. Sterling et al. BEOWULF: A Parallel Workstation for Scientific Computation. In *Proc. ICPP'95*, pages I:11–14, Oconomowoc, USA, August 1995.
20. R. Taniguchi et al. Software Platform for Parallel Image Processing and Computer Vision. In *Proc. SPIE*, volume 3166, pages 2–10, San Diego, USA, July 1997.
21. S.H. Unger. A Computer Oriented towards Spatial Problems. *Proceedings of the Institute of Radio Engineers*, 46:1744–1750, 1958.
22. J. Vendrig. *Interactive Exploration of Visual Content*. PhD thesis, ISIS, Faculty of Science, University of Amsterdam, The Netherlands, October 2002.
23. A.G. Webb. *Introduction to Biomedical Imaging*. Wiley-IEEE Press, 2002.