User Transparent Parallel Processing of the 2004 NIST TRECVID Data Set

F.J. Seinstra, C.G.M. Snoek, D. Koelma, J.M. Geusebroek, and M. Worring
ISLA, Informatics Institute, University of Amsterdam,
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
{fjseins, cgmsnoek, koelma, mark, worring}@science.uva.nl

Abstract

The Parallel-Horus framework, developed at the University of Amsterdam, is a unique software architecture that allows non-expert parallel programmers to develop fully sequential multimedia applications for efficient execution on homogeneous Beowulf-type commodity clusters. Previously obtained results for realistic, but relatively small-sized applications have shown the feasibility of the Parallel-Horus approach, with parallel performance consistently being found to be optimal with respect to the abstraction level of message passing programs. In this paper we discuss the most serious challenge Parallel-Horus has had to deal with so far: the processing of over 184 hours of video included in the 2004 NIST TRECVID evaluation, i.e. the de facto international standard benchmark for content-based video retrieval. Our results and experiences confirm that Parallel-Horus is a very powerful support-tool for state-of-the-art research and applications in multimedia processing.

1. Introduction

Even though many multimedia applications are ideally suited for parallel implementation [9, 10, 12, 14, 27, 29, 32], most researchers in multimedia do not benefit from the many commodity clusters available around the world today. This problem is primarily due to the lack of tools that can effectively help non-expert parallel programmers in the creation of high performance multimedia applications [18, 22]. Existing tools (e.g. message passing libraries) generally require non-expert users to identify the available parallelism at a level of detail that is beyond their skills. For this reason we have recognized that the key to effective high performance multimedia application development by non-experts in parallel computing lies in the availability of a familiar sequential (i.e. user transparent) programming model [22].

Parallel-Horus, a research and development project undertaken at the University of Amsterdam [21, 22, 23, 24], is a unique attempt to allow non-expert parallel programmers to develop *fully sequential* multimedia applications for efficient execution on Beowulf-type clusters. To enhance sustainability of the developed framework without compromising on the efficiency of execution, focus of the project is on the integration of algorithmic patterns for parallel image and video processing [24], automatic parallelization and optimization [23], and domain specific performance modeling [21, 24]. Earlier results have shown the feasibility of the Parallel-Horus approach, with parallel performance consistently being found to be optimal with respect to the abstraction level of message passing programs [22, 23].

This paper discusses the most serious problem Parallel-Horus has had to deal with so far: i.e. the processing of over 184 hours of video included in the 2004 NIST TRECVID data set [16, 25, 26]. TRECVID is the de facto international standard benchmark for content-based video retrieval, which aims at evaluating approaches to finding semantic concepts (e.g. roads, airplanes, etc) in archives of digitized news episodes from (e.g.) ABC and CNN. While we estimate that the processing of the entire data set would have taken around 250 days on the fastest sequential machine at our disposal, application of Parallel-Horus in combination with a distributed set of Beowulf-type commodity clusters significantly reduced the processing time to less than 60 hours. These performance gains were obtained without any parallelization effort whatsoever, thus playing an important role in the realization of our top-ranking TRECVID results.

This paper is organized as follows. Section 2 introduces the Parallel-Horus framework. In Section 3 the TRECVID evaluation is explained. Section 4 discusses our sequential solution to the semantic concept detection problem. The automatic parallelization of the developed application is discussed in Section 5. Section 6 presents an evaluation of results. Section 7 compares with related work. Future work is discussed in Section 8. Conclusions are given in Section 9.

2 Parallel-Horus: A Brief Overview

The core of the Parallel-Horus architecture consists of an extensive software library of data types and associated operations commonly applied in multimedia processing. To match the architecture's programming model with the expertise of the multimedia processing community, the library's application programming interface is made identical to that of an existing sequential library: Horus [11]. More specifically, rather than implementing a completely new library from scratch, the parallel functionality is integrated with the Horus implementation, such that all existing *sequential* code remains intact. Apart from reducing the required parallel implementation effort, this approach has the major advantage that the important properties of the Horus library (i.e., maintainability, extensibility, and portability) to a large extent transfer to Parallel-Horus as well.

Similar to other frameworks described in the literature [6, 14], the sequential Horus library is based on the abstractions of Image Algebra [20], a mathematical notation for specifying image and video processing algorithms. Image Algebra is an important basis for the design of a maintainable and extensible multimedia processing library, as it recognizes that a small set of *algorithmic patterns* can be identified that covers the bulk of all commonly applied functionality. In Horus each such pattern is implemented as a *generic algorithm*, using the C++ *function template mechanism* [28]. Any operation that maps onto the functionality as provided by such algorithm is obtained by instantiating the generic algorithm with the proper parameters, including the function to be applied to the individual data elements. For years the following set of generic algorithms has been available in Horus: (1) *unary pixel operation*, e.g. negation, absolute value, (2) *binary pixel operation*, e.g. addition, threshold, (3) *global reduction*, e.g. sum, maximum, (4) *neighborhood operation*, e.g. percentile, median, (5) *generalized convolution*, e.g. erosion, gauss, and (6) *geometric transformations*, e.g. rotation, scaling. Recently, additional and commonly used generic algorithms have been be introduced, including iterative and recursive neighborhood operations.

To ensure portability of Parallel-Horus to Beowulf-type commodity clusters, and to have full control over all communication behavior (and thus: efficiency), all parallel extensions have been implemented using MPI [13]. Also, to sustain a high maintainability level, each parallel operation contained in the library is implemented simply by concatenating data communication routines with fully sequential code blocks that are separately available in Horus. In this manner the source code for each sequential generic algorithm is fully reused in the implementation of its parallel counterpart, thus avoiding unnecessary code redundancy as much as possible [24].

For highest efficiency of full applications we have implemented all parallel operations such that they are capable of adapting to the performance characteristics of the parallel machine at hand, i.e. by incorporating run-time flexibility in the partitioning of data structures [21, 24]. More importantly, we have realized that it is not sufficient to consider parallelization of library operations *in isolation*. Therefore, we have extended the library with a surprisingly simple, but very efficient, run-time system for communication minimization [23]. This system ensures that the actual parallel code that is being executed is 1) *efficient*, often comparable to optimal hand-coded implementations, 2) *legal*, such that the program is deterministic and can never end in deadlock, and 3) *correct*, such that it produces output identical to the original sequential program. Hence, the system automatically parallelizes a *fully sequential* program *at runtime* by inserting communication primitives and additional memory management operations whenever necessary. This approach, referred to as *lazy parallelization*, is based on a simple finite state machine (fsm) specification. One of two fsm ingredients is a set of states, each corresponding to a valid internal representation of a distributed data structure at runtime. The other is a set of state transition functions, each of which defines how a valid data structure representation is transformed into another valid representation. A detailed description of the finite state machine, as well as an explanation of the fsm-based parallelization and optimization process, can be found in [23]. Finally, it is important to note that extensive experiments have shown that the Parallel-Horus system delivers close-to-optimal parallel performance for many state-of-the-art multimedia applications [22],

3 The 2004 NIST TRECVID Evaluation

TREC is a conference series sponsored by the National Institute of Standards and Technology (NIST) with additional support from other U.S. government agencies. The goal is to encourage research in information retrieval by providing a large test collection, uniform scoring procedures, and a forum for organizations interested in comparing their results. Since 2003 an independent evaluation track is being organized, devoted to research in automatic segmentation, indexing, and content-based retrieval of digital video streams: TRECVID [16].

The 2004 NIST TRECVID evaluation defines four main tasks, at least one of which must be completed to participate in the evaluation. The tasks are as follows:

- shot boundary determination
- story segmentation
- high-level feature extraction
- search.

The University of Amsterdam participated in TRECVID 2004 by completing the feature extraction task, as well as

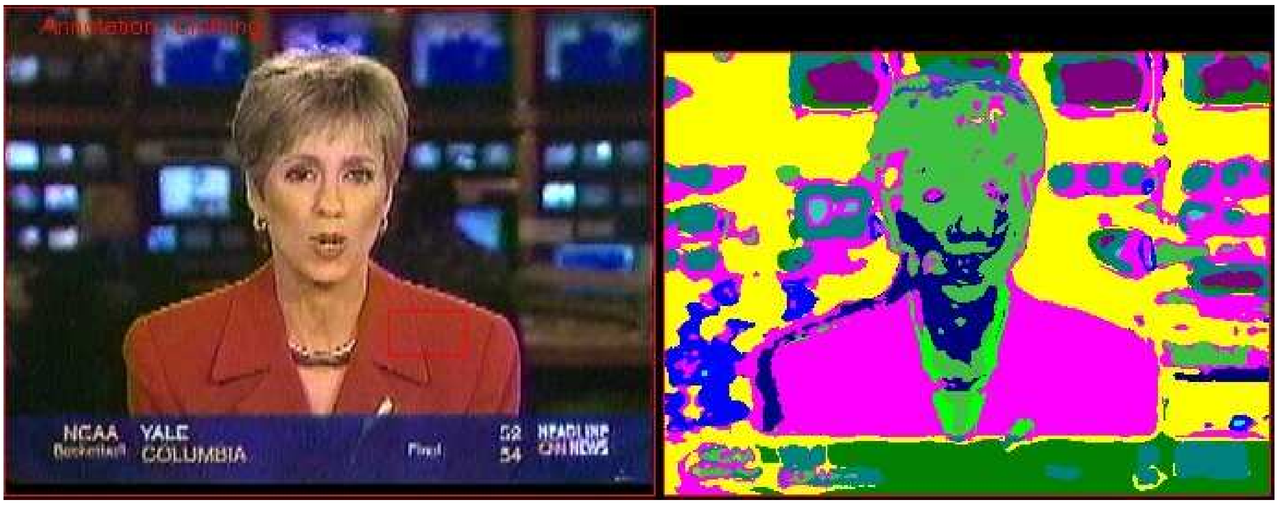

Figure 1. *Example frame from the 2004 NIST TRECVID data set (left) and labeled segmentation results (right).*

the search task. In the remainder of this paper we will focus on the feature extraction task only.

The feature extraction task was defined as follows [16]: Given the 2004 NIST TRECVID video data set, a common shot boundary reference for this data set, and a list of feature definitions (see below), participants must return for each feature a list of at most 2000 shots from the data set, ranked according to the highest possibility of detecting the presence of that feature. Each feature is assumed to be binary, i.e. it is either present or absent in a shot.

The 2004 NIST TRECVID video data set consisted of over 184 hours of digitized news episodes from ABC and CNN, an example of which is shown in the left half of Figure 1. In addition, ten feature definitions were given:

- **Boat/ship**: segment contains video of at least one boat, canoe, kayak, or ship of any type.
- **Bill Clinton**: segment contains video of Bill Clinton.
- **Madeleine Albright**: segment contains video of Madeleine Albright.
- **Train**: segment contains video of one or more trains, or railroad cars which are part of a train.
- **Beach**: segment contains video of a beach with the water and the shore visible.
- **Airplane takeoff**: segment contains video of an airplane taking off, moving away from the viewer.
- **People walking/running**: segment contains video of more than one person walking or running.
- **Physical violence**: segment contains video of violent interaction between people and/or objects.
- **Road**: segment contains video of part of a road, any size, paved or not.
- **Basket scored**: segment contains video of a basketball passing down through the hoop and into the net to score a basket — as part of a game or not.

4 Generic Semantic Concept Detection

Our approach to the feature extraction problem is based on the so-called Semantic Value Chain (SVC), a novel method for generic semantic concept detection in multimodal video repositories [26]. The SVC extracts semantic concepts from video based on three consecutive analysis links, i.e. the Content Link, the Style Link, and the Semantic Context Link. The Content Link works on the video data itself, whereas the Style Link and the Semantic Context Link work on higher level semantic representations.

In the Content Link we view video documents from the data perspective. In general, three modalities can be identified in video documents, i.e. the auditory, textual, and visual modality. In our approach, detectors are first applied to individual modalities. The results are then fused into an integrated Content Link detector. Based on validation experiments the best-off hypothesis for a single concept serves as the input for the next link. In the remainder of this paper we focus on the processing of the visual modality, as this is by far the most time-consuming part of the complete system.

```
inputVideo = openFile(videoFileName);
semanticConcepts = readFile(conceptsFileName);
svMachine = initSupportVectorMachine(semanticConcepts);
WHILE (NOT endOfVideo(inputVideo)) DO
  inputFrame = getNextFrame(inputVideo, skipFrames = 15);
  invFeatureVector = buildInvariantFeatureVector(inputFrame);
  labeledFrame = doSVMlabeling(invFeatureVector, svMachine);
  rankHistogram = getHistogram(labeledFrame);
  writeFile(rankHistogram);
OD
```

Listing 1. *Pseudo code for SVM-based visual analysis.*

4.1 Visual Analysis

The visual modality is analyzed at the image (or video frame) level. Our approach is presented in highly simplified pseudo code in Listing 1. After obtaining video data from file, for each 15th video frame visual features are extracted by using Gaussian color invariant measurements [7]. RGB color values are decorrelated by transformation to an opponent color system. Successively, acquisition and compression noise is suppressed by Gaussian smoothing. By varying the size of the Gaussian filters $\sigma = \{1, 2, 3.5\}$ a color representation is obtained consistent with variations in target object size. Global and local intensity variations are suppressed by normalizing each color value by its intensity, resulting in two chromaticity values per color pixel. Furthermore, rotationally invariant features are obtained by taking Gaussian derivative filters, and combining the responses into two chromatic gradient magnitude measures. These seven features, calculated over three scales, yield a combined 21-dimensional feature vector per pixel. This sequence of operations is represented in Listing 1 by a single call to `buildInvariantFeatureVector`. It should be noted that this single operation boils down to a sequence of unary and binary pixel operations, and generalized convolutions, as referred to in Section 2.

The obtained invariant feature vector serves as the input for a multi-class Support Vector Machine [4, 30] that associates each pixel to one of the predefined regional visual concepts. In Listing 1 this is performed by the `doSVMlabeling` operation. As before, this operation represents a sequence of unary and binary pixel operations. The SVM labeling results in a weak semantic segmentation of a video frame in terms of regional visual concepts (see Figure 1 for an example). This result is written out to file in condensed format (i.e.: a histogram) for subsequent processing.

The described segmentation of video frames into regional visual concepts at the granularity of a pixel is computationally intensive. This is especially so if one aims to analyze as many frames as possible. In our approach the visual analysis of a single video frame requires around 16 seconds on the fastest sequential machine at our disposal (i.e.: a 3.2 GHz Pentium IV with 1 GByte of RAM). Consequently, when processing two frames per second at a frame rate of 30, the required processing time for the entire TRECVID data set would be around 250 days. Clearly, for the initial visual analysis phase of our complete system parallel execution is highly desired.

5 Parallel Execution

As Parallel-Horus shields all parallelization issues from the user, the sequential pseudo code of Listing 1 directly constitutes a program that can be executed on a Beowulf cluster as well. Performance optimization and communication minimization are taken care of by the integrated finite state machine-based run-time system [23]. Here we should note that, as of yet, we can not make publicly available the actual code of our TRECVID application. However, an initial proof-of-concept implementation (in `C`) of the Parallel-Horus framework, as well as several small-sized example applications, are freely available for inspection at http://www.science.uva.nl/˜fjseins/ParHorusCode/.

For this particular sequential application, the resulting parallel execution involves only three different communication steps per video frame. Initially, each input frame obtained from file is scattered throughout the parallel system (note: our architecture does not support parallel I/O). Next, in the generation of the invariant feature vector, the required generalized convolutions (e.g. Gaussian smoothing) cause neighboring nodes in the logical CPU grid to exchange their image borders (or *shadow regions*). In all cases, the extent of the border in each of the image's dimensions is half the size of the Gaussian kernel in that dimension minus one pixel. Finally, before the resulting histogram is written out to file, all partial histograms are gathered to a single processing unit. Apart from these communication operations, all processing units run fully independently, in a data parallel manner. As such, the program executes in exactly the same way as would have been the case for a hand-coded data parallel version implemented in MPI. In other words, the resulting parallel execution is optimal with respect to the abstraction level of message passing programs.

6 Results

In this section we give an evaluation of the results obtained by applying Parallel-Horus in the 2004 NIST TRECVID evaluation. First we will discuss our view on the importance of Parallel-Horus in the overall feature detection results generated by the University of Amsterdam (UvA). This is followed by a discussion of the parallel performance obtained automatically with Parallel-Horus.

Feature	UvA Rank	Nr. Participants
People walking	1st	7
Physical violence	1st	6
Boat/ship	2nd	7
M. Albright	2nd	8
B. Clinton	2nd	11
Airplane takeoff	2nd	7
Road	2nd	6
Train	3rd	6
Beach	7th	9
Basket scored	8th	10

Table 1. *Per feature ranking of the UvA system.*

6.1 Application of Parallel-Horus in TRECVID

The 2004 NIST TRECVID feature extraction task was completed by several strong international competitors, including IBM Research [1], and the Informedia team from Carnegie Mellon University [8]. The TRECVID feature extraction task was an enormous challenge, which is reflected in the fact that there were not more than 11 participating teams. Moreover, in this field of 11 teams, only 6 teams were capable of presenting results for the complete set of 10 predefined features (see Section 3). First, this is because the detection of all 10 features in the entire TRECVID data set was by far too time-consuming for many participating teams. In addition, this is because the development of parallel solutions to the TRECVID problem would require expertise beyond the means of most participants.

The importance of applying Parallel-Horus in the UvA TRECVID system can be seen from the fact that we have been able to present results for all 10 predefined features. In addition, the performance gains obtained from parallel execution clearly played an important role in the realization of our overall top-ranking results (see Table 1). Despite the fact that Parallel-Horus was applied in the visual analysis phase of the complete system only, it had a positive effect on our TRECVID participation as a whole. While all functionality was developed in a fully sequential manner, parallel performance was obtained entirely for free — thus allowing for a more thorough comparison of alternative algorithmic approaches in the research phase, for more accurate parameter tuning in the development phase, and for more in-depth scene analysis in the final processing phase.

6.2 Performance Evaluation

The visual analysis phase of our TRECVID system has been executed on the Distributed ASCI Supercomputer 2 (DAS-2), a wide-area distributed computer located at five different universities in The Netherlands [3]. DAS-2 consists of five Beowulf-type clusters, one of which contains 72 nodes, and four of which have 32 nodes (200 nodes in total). All nodes consist of two 1.0 GHz Pentium III CPUs, at least 1.0 GByte of RAM, and are connected by a Myrinet-2000 network. At the time of measurement, the nodes ran the RedHat Linux 7.2 operating system. The Parallel-Horus architecture was compiled using `gcc` 3.2.2 (at highest level of optimization) and linked with MPICH-GM, which uses Myricom's GM as its message passing layer on Myrinet.

Figure 2 presents measurements for the processing of a single video frame from the TRECVID data set. As the DAS-2 system is used for other research projects as well, results are given here for a system of up to 64 CPUs only. Also, it should be noted that these results present a lower bound on the obtainable speedup for this application, as the size of each video frame is small: 352×240 pixels (MPEG-1 standard). In future TRECVID evaluations video streams are expected to be encoded in the MPEG-2 format (with frame-sizes of 720×576 pixels), resulting in a much higher speedup potential per frame (as proven in [22] for several other applications having similar parallel behavior).

From Figure 2 we can see that the execution time per frame drops from more than 27 seconds on a single DAS-2 node to slightly over 1.5 seconds using 64 processing units. This gives a speedup of 17.84 with an efficiency of 27.88 on the maximum number of CPUs used in this evaluation. These speedup figures were of great importance in the research and development phases of the UvA system, as these allowed immediate evaluation of different algorithms, and a thorough tuning of algorithmic parameters.

In the final processing phase we manually applied all five clusters of the DAS-2 system in parallel, executing multiple runs on each individual cluster in parallel as well. To have a good trade-off between the number of parallel tasks that needed to be supervised and managed[1], and the obtained speedup as discussed above, each individual task (i.e. the processing of a single video) was given a total of 16 dual CPUs. Given the execution time of around 27 seconds for a single frame as given in Figure 2, we estimate that the processing of the whole TRECVID data set would have taken more than one year on a single DAS-2 node (and around 250 days on the fastest sequential machine at our disposal). Application of Parallel-Horus in combination with the five DAS-2 clusters significantly reduced the pure processing time to less than 60 hours, resulting in an overall speedup of around 172 on the complete DAS-2 system. We would like to stress again that these results were obtained without any parallel programming effort whatsoever.

[1]the TRECVID data set could not be made accessible in its entirety at once, thus requiring manual copying of video data and results to and from the individual DAS-2 clusters

# CPUs	time (s)
1	27.443
2	12.883
4	7.145
8	4.777
16	3.388
24	2.685
32	2.223
40	1.901
48	1.715
56	1.588
64	1.538

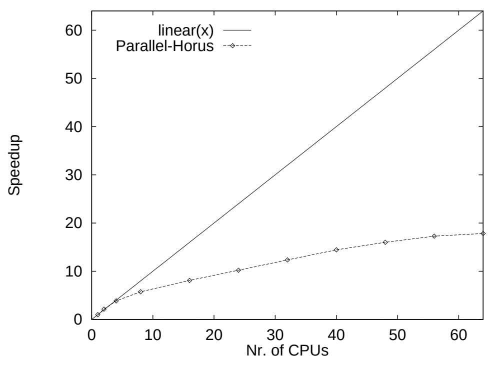

Figure 2. *Performance (left) and speedup characteristics (right) for the SVM-based visual analysis, for a single video frame of size 352×240 pixels (MPEG-1 video data).*

7 Comparison with Related Work

In [22] we have made a performance comparison between the Parallel-Horus architecture and several related tools described in the literature. The comparison is based on a well-known stereo vision application [17] which — in its parallel behavior — is comparable to the visual analysis phase of our TRECVID system. The following briefly presents the main results of the comparison.

First, a comparison is made with results obtained for the stereo vision application written in a specialized parallel programming language (SPAR [19]), which was executed on a single DAS-2 cluster. Also, the codes generated by the SPAR front-end and that of Parallel-Horus were compiled in an identical manner. Measurements showed Parallel-Horus to provide superior sequential performance of about a factor 5, and better speedup — clearly indicating that the overhead the Parallel-Horus approach is much smaller than that of the SPAR run-time system.

Second, a comparison is made with results obtained for an implementation in the Adapt parallel image processing language [31]. A true comparison with this work turned out to be difficult, however, as the results were obtained on a significantly different machine (i.e., a set of iWarp processors, with a better potential for obtaining high speedup than the DAS-2). Even so, our software architecture showed superior performance (of about a factor 2) with comparable speedup characteristics over a large range of CPUs.

Most interesting, however, is the comparison with *Easy-PIPE* [15], a library-based framework for parallel image processing similar to Parallel-Horus. The most distinctive feature of this architecture is that it incorporates a mechanism for combining data and task parallelism. Also, *Easy-PIPE* does not shield *all* parallelism from the application programmer. As a consequence from these differences, *Easy-PIPE* has the potential of outperforming our architecture, which is fully user transparent, and strictly data parallel. However, performance and speedup characteristics for the stereo vision application obtained on the DAS-2 architecture show that our implementations far better exploit the available parallelism than *Easy-PIPE*. Part of the difference is accounted for by the fact that *Easy-PIPE* does not incorporate an explicit optimization mechanism for removal of redundant communication steps. In addition, the run time parallelization overhead of *Easy-PIPE* turned out to be much higher than that of Parallel-Horus.

8 Future Work

Parallel-Horus is an ongoing research and development project. To enhance the user's expressiveness, while preserving access to the available multimedia functionality, we are currently investigating alternative sequential programming models. A promising starting point is our initial study of functional programming models for multimedia processing, in which the OCaml language [5] was identified as a good candidate for interfacing with a Parallel-Horus backend [2]. Also, we are investigating alternative execution scenarios that shield the user from platform specific peculiarities related to the actual running of Parallel-Horus programs. One desirable execution scenario is to have a multimedia server running on a distributed set of (potentially heterogeneous) resources, such that the available *Multimedia Services* can be called directly from within a sequential program running on a local machine. Finally, due to the emerging need for the processing of large image and video archives, future multimedia applications have a higher degree of inherent task parallelism than before. Applying an RPC-style execution model based on Multimedia Services allows for a traditional sequential programming model, while making possible the exploitation of integrated data and task parallelism, i.e. by asynchronous (task) parallel invocation of services, each of which may run in data parallel fashion. We expect the Parallel-Horus architecture to move in this direction in the near future.

9 Conclusions

Parallel-Horus is a unique software architecture that allows non-expert parallel programmers to develop fully sequential multimedia applications for efficient execution on Beowulf-type commodity clusters. In this paper we have described the application of Parallel-Horus in the low-level processing of over 184 hours of video included in the 2004 NIST TRECVID evaluation, i.e. the de facto international standard benchmark for content-based video retrieval. While we estimate that the processing of the entire data set would have taken around 250 days on the fastest sequential machine at our disposal, application of Parallel-Horus in combination with the 200-node Distributed ASCI Supercomputer reduced the processing time to around 2.5 days. We have indicated that these performance gains were obtained without any parallel programming effort whatsoever, thus playing an important role in the realization of our top-ranking TRECVID results in a field of very strong international competitors (a.o. including IBM Research and Carnegie Mellon University).

The application of Parallel-Horus in the TRECVID evaluation proved to be very significant. Most importantly, it allowed us to thoroughly process the enormous 2004 NIST TRECVID video data collection virtually without effort. It enabled us to develop all feature detection functionality in a sequential manner, and to make full advantage of the automatically provided parallel performance. As a result, it allowed for a more thorough comparison of alternative algorithmic approaches in the research phase, for more accurate parameter tuning in the development phase, and for more extensive scene analysis in the final processing phase.

In conclusion: irrespective of the need for more research as described above, our results and experiences confirm that the current Parallel-Horus implementation is a very powerful support-tool for state-of-the-art research and applications in multimedia processing. As such, Parallel-Horus has an immediate and stimulating effect on the study of the many computationally demanding problems in multimedia processing. The described NIST TRECVID evaluation is just one of these.

10 Acknowledgements

The authors would like to thank Kees Verstoep (Vrije Universiteit, Amsterdam) for his support in setting up and employing the Distributed ASCI Supercomputer 2.

References

[1] A. Amir, M. Berg, S. Chang, G. Iyengar, C.-Y. Lin, A. Natsev, C. Neti, H. Nock, M. Naphade, W. Hsu, J. Smith, B. Tseng, Y. Wu, and D. Zhang. IBM Research TRECVID-2003 Video Retrieval System. In *Proceedings of the TRECVID 2003 Workshop*, Nov. 2003.

[2] A. Bagdanov. *Style Characterization of Machine Printed Texts*. PhD thesis, Faculty of Science, University of Amsterdam, The Netherlands, May 2004.

[3] H. Bal et al. The Distributed ASCI Supercomputer Project. *Operating Systems Review*, 34(4):76–96, Oct. 2000.

[4] C.-C. Chang and C.-J. Lin. *LIBSVM: a Library for Support Vector Machines*, 2001. Software available at `www.csie.ntu.edu.tw/~cjlin/libsvm/`.

[5] G. Cousineau and M. Mauny. *The Functional Approach to Programming*. Cambridge University Press, 1998.

[6] D. Crookes, P. Morrow, and P. McParland. IAL: A Parallel Image Processing Programming Language. *IEE Proceedings, Part I*, 137(3):176–182, June 1990.

[7] J. Geusebroek, R. van den Boomgaard, A. Smeulders, and H. Geerts. Color Invariance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(12):1338–1350, 2001.

[8] A. Hauptmann, R. Baron, M.-Y. Chen, M. Christel, P. Duygulu, C. Huang, R. Jin, W.-H.Lin, T. Ng, N. Moraveji, N. Papernick, C. Snoek, G. Tzanetakis, J. Yang, R. Yang, and H. Wactlar. Informedia at TRECVID 2003: Analyzing and Searching Broadcast News Video. In *Proceedings of the TRECVID 2003 Workshop*, Nov. 2003.

[9] L. Jamieson, E. Delp, C.-C. Wang, J. Li, and F. Weil. A Software Environment for Parallel Computer Vision. *IEEE Computer*, 25(2):73–75, Feb. 1992.

[10] Z. Juhasz and D. Crookes. A PVM Implementation of a Portable Parallel Image Processing Library. In *Parallel Virtual Machine - EuroPVM'96, Third European PVM Conference*, pages 188–196, Munich, Germany, Oct. 1996.

[11] D. Koelma et al. Horus C++ Reference, Version 1.1. Technical report, ISIS, Faculty of Science, University of Amsterdam, The Netherlands, Jan. 2002.

[12] C. Lee and M. Hamdi. Parallel Image Processing Applications on a Network of Workstations. *Parallel Computing*, 21(1):137–160, Jan. 1995.

[13] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard (version 1.1). Technical report, University of Tennessee, Knoxville, Tennessee, June 1995. Available at `http://www.mpi-forum.org`.

[14] P. Morrow, D. Crookes, J. Brown, G. McAleese, D. Roantree, and I. Spence. Efficient Implementation of a Portable Parallel Programming Model for Image Processing. *Concurrency: Pract. & Exp.*, 11:671–685, Sept. 1999.

[15] C. Nicolescu and P. Jonker. EASY-PIPE - An Easy to Use Parallel Image Processing Environment Based on Algorithmic Skeletons. In *Proceedings of the 15th International Parallel & Distributed Processing Symposium - Workshop on Parallel and Distributed Computing in Image Processing, Video Processing, and Multimedia*, San Francisco, California, USA, Apr. 2001.

[16] NIST. TREC Video Retrieval Evaluation Homepage. URL: http://www-nlpir.nist.gov/projects/trecvid/, Oct. 2004.

[17] M. Okutomi and T. Kanade. A Multiple-Baseline Stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(4):353–363, Apr. 1993.

[18] C. Pancake and D. Bergmark. Do Parallel Languages Respond to the Needs of Scientific Programmers? *IEEE Computer*, 23(12):13–23, Dec. 1990.

[19] C. Reeuwijk, A. van Gemund, and H. Sips. Spar: A Programming Language for Semi-Automatic Compilation of Parallel Programs. *Concurrency: Practice and Experience*, 9(11):1193–1205, Nov. 1997.

[20] G. Ritter and J. Wilson. *Handbook of Computer Vision Algorithms in Image Algebra*. CRC Press, Inc, 1996.

[21] F. 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.

[22] F. 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.

[23] F. Seinstra, D. Koelma, and A. 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*, 15(10):865–877, Oct. 2004.

[24] F. Seinstra, D. Koelma, and J. Geusebroek. A Software Architecture for User Transparent Parallel Image Processing. *Parallel Computing*, 28(7–8):967–993, Aug. 2002.

[25] C. Snoek and M. Worring. Multimodal Video Indexing: A Review of the State-of-the-art. *Multimedia Tools and Applications*, 2005. (in press).

[26] C. Snoek, M. Worring, J. Geusebroek, D. Koelma, and F. Seinstra. The MediaMill TRECVID 2004 Semantic Video Search Engine. In *Proceedings of the TRECVID 2004 Workshop*, Nov. 2004.

[27] J. Squyres, A. Lumsdaine, and R. Stevenson. A Toolkit for Parallel Image Processing. In *Parallel and Distributed Methods for Image Processing II, Proceedings of SPIE*, volume 3452, San Diego, California, USA, July 1998.

[28] B. Stroustrup. *The C++ Programming Language, 3rd Edition*. Addison-Wesley, 1997.

[29] R. Taniguchi, Y. Makiyama, N. Tsuruta, S. Yonemoto, and D. Arita. Software Platform for Parallel Image Processing and Computer Vision. In *Parallel and Distributed Methods for Image Processing, Proceedings of SPIE*, volume 3166, pages 2–10, San Diego, California, USA, July 1997.

[30] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, USA, 2th edition, 2000.

[31] J. Webb. Implementation and Performance of Fast Parallel Multi-Baseline Stereo Vision. In *Proceedings of the 1993 DARPA Image Understanding Workshop*, pages 1005–1010, Apr. 1993.

[32] H. Yoshimoto, D. Arita, and R. Taniguchi. Real-Time Image Processing on IEEE1394-based PC Cluster. In *Proceedings of the 15th International Parallel & Distributed Processing Symposium - Workshop on Parallel and Distributed Computing in Image Processing, Video Processing, and Multimedia*, San Francisco, USA, Apr. 2001.