

A Comparison of Distributed Data Parallel Multimedia Computing over Conventional and Optical Wide-Area Networks*

Fangbin Liu
ISLA, Informatics Institute
University of Amsterdam, Kruislaan 403
1098 SJ Amsterdam, The Netherlands
fliu@science.uva.nl

Frank J. Seinstra
Department of Computer Science
Vrije Universiteit, De Boelelaan 1081A
1081 HV Amsterdam, The Netherlands
fjseins@cs.vu.nl

Abstract

The research area of Multimedia Content Analysis (MMCA) considers all aspects of the automated extraction of knowledge from multimedia data streams and archives. As individual compute clusters can not satisfy the increasing computational demands of emerging MMCA problems, distributed supercomputing on collections of compute clusters is rapidly becoming indispensable. A well-known manner of obtaining speedups in MMCA is to apply data parallel approaches, in which commonly used data structures (e.g. video frames) are being scattered among the available compute nodes. Such approaches work well for individual compute clusters, but - due to the inherently large wide-area communication overheads - these are generally not applied in distributed cluster systems. Given the increasing availability of low-latency, high-bandwidth optical wide-area networks, however, wide-area data parallel execution may now become a feasible acceleration approach. This paper discusses the wide-area data parallel execution of a realistic MMCA problem. It presents experimental results obtained on real distributed systems, and provides a feasibility analysis of the applied parallelization approach.

1 Introduction

Image and video data is rapidly gaining importance along with recent deployment of publicly accessible digital television archives, surveillance cameras in public locations, and automatic comparison of forensic video evidence. In a few years, analyzing the content of such data will be a problem of phenomenal proportions, as digital video may produce high data rates, and multimedia archives steadily run into Petabytes of storage space. Consequently, for urgent problems in multimedia content analysis, distributed supercomputing on large collections of clusters (Grids) is rapidly becoming indispensable.

It is well-known that data parallel approaches in image, video, and multimedia computing can provide efficient acceleration. In such approaches images and video frames are scattered among the available compute nodes, such that each node calculates over a partial structure only. Inter-node dependencies are then resolved by communication between the nodes. In the past 15 years numerous multimedia applications have been executed successfully in this manner, in particular using compute clusters [5], [8], [11].

For fine grain data parallel execution, inter-node communication is the major hurdle for obtaining efficient speedups. Because communication *within* a cluster is generally an order of magnitude faster than communication *between* clusters, data parallel execution *using multiple cluster systems* is generally believed to be inefficient. However, with the advent of low-latency, high-bandwidth optical networks, wide-area data parallel execution may now become a feasible acceleration approach. Investigation of this approach is specifically relevant for applications that work on very large data structures (e.g. hyperspectral images).

In this paper we apply wide-area data parallelism to a low-level problem in multimedia content analysis, i.e. curvilinear structure detection. For the development of the application we use a multimedia computing library (Parallel-Horus [11]) that allows programmers to implement data parallel applications as fully sequential programs. The application is tested on two distributed multi-cluster systems. One system is equipped with a dedicated wide-area optical interconnect, while the other uses conventional Internet links. We present experimental results, and provide a feasibility analysis of the applied parallelization approach.

This paper is organized as follows. Section 2 introduces the Parallel-Horus library. Section 3 describes our two experimental setups. Section 4 describes the low level image processing problem as applied in our experiments. Subsequently, Section 5 gives an evaluation of the performance and speedup results obtained on the two distributed platforms. Concluding remarks are given in Section 6.

2 Parallel-Horus

Parallel-Horus [11] is a cluster programming framework, implemented in C++ and MPI, that allows programmers to implement data parallel multimedia applications as fully sequential programs. The library's API is made identical to that of an existing sequential library: Horus [4]. Similar to other frameworks [7], Horus recognizes that a small set of *algorithmic patterns* can be identified that covers the bulk of all commonly applied functionality.

Parallel-Horus includes patterns for functionality such as unary and binary pixel operations, global reduction, neighborhood operation, generalized convolution, and geometric transformations. Current developments include patterns for operations on large datasets, as well as patterns on increasingly important derived data structures, such as feature vectors. For reasons of efficiency, all Parallel-Horus operations are capable of adapting to the performance characteristics of a parallel machine at hand, i.e. by being flexible in the partitioning of data structures. Moreover, it was realized that it is not sufficient to consider parallelization of library operations *in isolation*. Therefore, the library was extended with a run-time approach for communication minimization (called *lazy parallelization*) that automatically parallelizes a fully sequential program at runtime by inserting communication primitives and additional memory management operations whenever necessary [9].

Results for realistic multimedia applications have shown the feasibility of the Parallel-Horus approach, with data parallel performance (obtained on individual cluster systems) consistently being found to be optimal with respect to the abstraction level of message passing programs [11]. Notably, Parallel-Horus was applied in the 2004, 2005, and 2006 NIST TRECVID benchmark evaluations for content-based video retrieval, and played a crucial role in achieving top-ranking results in a field of strong international competitors [11], [12]. Moreover, recent extensions to Parallel-Horus, that allow for services-based multimedia Grid computing, have been applied successfully in large-scale distributed systems, involving hundreds of massively communicating compute resources covering our entire globe [11]. Clearly, Parallel-Horus is a system that serves well in bringing the benefits of high-performance computing to the multimedia community, but we are constantly working on further improvements, as exemplified in the following sections.

3 Experimentation Platforms

In the next sections we evaluate the obtained performance gains for distributed data parallel execution of a fine-grained Parallel-Horus program. To this end, the next section provides a description of the multimedia application, and the applied parallelization approaches. This section discusses the hardware platforms applied in the experiments.

3.1 DAS-2

The first platform is the (old) Distributed ASCII Supercomputer 2 (DAS-2), a 200-node system located at five different universities in The Netherlands: Vrije Universiteit Amsterdam (72 nodes), Leiden University, University of Amsterdam, Delft University of Technology, and University of Utrecht (32 nodes each). All nodes consist of two 1-GHz Pentium-III CPUs with up to 2 GByte of RAM, and are connected by a Myrinet-2000 network. At the time of measurement, the nodes ran RedHat Enterprise Linux AS 3.2.3. The cluster at the University of Amsterdam has been dismantled in 2007, leaving a four-cluster system of 168 nodes.

3.2 DAS-3 and StarPlane

The second platform is the recently installed follow-up to DAS-2, i.e. the DAS-3, see <http://www.cs.vu.nl/das3/>. Similar to DAS-2, DAS-3 is a five-cluster system, with the following specifications: Vrije Universiteit Amsterdam (85x4 cores, 2.4 GHz), Leiden University (32x2 cores 2.6 GHz), University of Amsterdam (1 cluster of 40x4 cores, 2.2 GHz; 1 cluster of 46x2 cores, 2.4 GHz), and Delft University of Technology (68x2 cores, 2.4 GHz). Besides the ubiquitous 1 and 10 GBit/s Ethernet networks, DAS-3 incorporates the recent high-speed Myri-10G interconnect technology from Myricom as an internal high-speed interconnect.

DAS-3 employs a novel 10Gbit/s wide-area interconnect based on light paths (StarPlane, see www.starplane.org). The rationale of the StarPlane optical interconnect is to allow part of the photonic network infrastructure of the Dutch SURFnet6 network to be manipulated by Grid applications to optimize performance. The novelty of StarPlane is that it does give flexibility directly to applications by allowing them to choose one of multiple logical network topologies in real time, ultimately with subsecond switching times.

4 Line Detection

As discussed in [1], the important problem of detecting lines and linear structures in images is solved by considering the second order directional derivative in the gradient direction, for each possible line direction. This is achieved by applying anisotropic Gaussian filters, parameterized by orientation θ , smoothing scale σ_u in the line direction, and differentiation scale σ_v perpendicular to the line, given by

$$r''(x, y, \sigma_u, \sigma_v, \theta) = \sigma_u \sigma_v \left| f_{v v}^{\sigma_u, \sigma_v, \theta} \right| \frac{1}{b \sigma_u \sigma_v \theta}, \quad (1)$$

with b the line brightness. When the filter is correctly aligned with a line in the image, and σ_u, σ_v are optimally tuned to capture the line, filter response is maximal. Hence, the per pixel maximum line contrast over the filter parameters yields line detection:

$$R(x, y) = \arg \max_{\sigma_u, \sigma_v, \theta} r''(x, y, \sigma_u, \sigma_v, \theta). \quad (2)$$

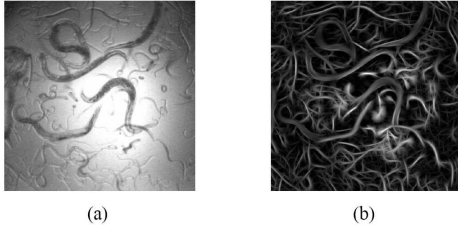


Figure 1. Detection of *C. Elegans* worms (courtesy of Janssen Pharmaceuticals, Belgium).

Figure 1(a) gives a typical example of an image used as input to this algorithm. Results obtained for a reasonably large subspace of $(\sigma_u, \sigma_v, \theta)$ are shown in Figure 1(b).

4.1 Sequential Implementations

The line detection problem can be implemented sequentially in many ways. In the remainder of this section we consider two possible approaches. First, for each orientation θ it is possible to create a new filter based on σ_u and σ_v . In effect, this yields a rotation of the filters, while the orientation of the input image remains fixed. Hence, a sequential implementation based on this approach (which we refer to as *Conv2D*) implies full 2-dimensional convolution for each filter. A second approach (called *ConvRot*) is to keep the orientation of the filters fixed, and to rotate the input image instead. The filtering now proceeds in a two-stage separable Gaussian, applied along the x - and y -direction.

Pseudo code for the *ConvRot* algorithm is given in Listing 1. The program starts by rotating the input image for a given orientation θ . Then, for all (σ_u, σ_v) combinations the filtering is performed by xy -separable Gaussian filters. For each orientation step the maximum response is combined in a single contrast image structure. Finally, the temporary contrast image is rotated back to match the orientation of the input image, and the maximum response image is obtained.

For the *Conv2D* algorithm the pseudo code is given in Listing 2. Filtering is performed in the inner loop by a full

```

FOR all orientations  $\theta$  DO
  RotatedIm = GeometricOp(OriginalIm, "rotate",  $\theta$ );
  ContrastIm = UnPixOp(ContrastIm, "set", 0);
  FOR all smoothing scales  $\sigma_u$  DO
    FOR all differentiation scales  $\sigma_v$  DO
      FiltIm1 = GenConvOp(RotatedIm, "gaussXY",  $\sigma_u, \sigma_v, 2, 0$ );
      FiltIm2 = GenConvOp(RotatedIm, "gaussXY",  $\sigma_u, \sigma_v, 0, 0$ );
      DetectedIm = BinPixOp(FiltIm1, "absdiv", FiltIm2);
      DetectedIm = BinPixOp(DetectedIm, "mul",  $\sigma_u * \sigma_v$ );
      ContrastIm = BinPixOp(ContrastIm, "max", DetectedIm);
    OD
  OD
  BackRotatedIm = GeometricOp(ContrastIm, "rotate",  $-\theta$ );
  ResultIm = BinPixOp(ResultIm, "max", BackRotatedIm);
OD

```

Listing 1: Pseudo code for the *ConvRot* algorithm.

```

FOR all orientations  $\theta$  DO
  FOR all smoothing scales  $\sigma_u$  DO
    FOR all differentiation scales  $\sigma_v$  DO
      FiltIm1 = GenConvOp(OriginalIm, "gauss2D",  $\sigma_u, \sigma_v, 2, 0$ );
      FiltIm2 = GenConvOp(OriginalIm, "gauss2D",  $\sigma_u, \sigma_v, 0, 0$ );
      ContrastIm = BinPixOp(FiltIm1, "absdiv", FiltIm2);
      ContrastIm = BinPixOp(ContrastIm, "mul",  $\sigma_u * \sigma_v$ );
      ResultIm = BinPixOp(ResultIm, "max", ContrastIm);
    OD
  OD

```

Listing 2: Pseudo code for the *Conv2D* algorithms.

two-dimensional convolution. On a state-of-the-art sequential machine either version of our program may take from a few minutes up to several hours to complete, depending on the size of the input image and the extent of the chosen parameter subspace. Consequently, for the directional filtering problem parallel execution is highly desired.

4.2 Parallel Execution

Automatic parallelization of the *ConvRot* program has resulted in an optimal schedule for this application [10]. The *OriginalIm* structure is broadcast to all nodes before each calculates its partial *RotatedIm* structure. The broadcast takes place only once, as *OriginalIm* is not updated in any operation. Subsequently, all operations in the innermost loop are executed locally on partial image data. The only need for communication is in the exchange of image borders in the Gaussian convolution operations.

The two final operations in the outermost loop are executed in a data parallel manner as well. As this requires the distributed image *ContrastIm* to be available in full at each node [10], a gather-to-all operation is performed. Finally, a partial maximum response image *ResultIm* is calculated on each node, which requires a final gather operation to be executed just before termination of the program.

The schedule generated for the *Conv2D* program is more straightforward. First, the *OriginalIm* structure is scattered such that each node obtains an equal-sized partial image. Next, all operations are performed in parallel, with border exchange communication required in the convolution operations only. Finally, before termination of the program *ResultIm* is gathered to a single node.

4.2.1 Lazy vs. Naive Parallelization

The parallelization described above is based on the Parallel-Horus approach of 'lazy parallelization' that removes redundant communication steps at application run-time, with close to zero overhead. While this approach is essential for obtaining highest performance, we can decide not to apply lazy parallelization at all. If we do so, applications will communicate much more than needed, but will still produce the same result. Here, we refer to this latter approach as 'naive parallelization'. In the following, our 'naive' parallel runs are included only to enhance our understanding of the impact of communication on the obtained results.

5 Evaluation

To run our experiments on the DAS-2 system, we have compiled our programs with the MPICH-G2 library [3], which provides the correct communication setup for MPI programs consisting of multiple components that need to run on multiple sites. For highest performance, we have initialized the communication such that, apart from IP-based communication between cluster sites, we use the local Myrinet-2000 network for intra-cluster communication. To start our multi-cluster MPI-jobs on DAS-2, we have used the DRunner job submission tool, as part of the KOALA system [6], that allows for the simultaneous allocation of resources in multiple clusters.

As the KOALA system has not yet been put in place on the DAS-3 system, we have applied the OpenMPI library instead [2]. In our case, the relevant feature of OpenMPI is its support for heterogeneous networks. This is provided in a manner which is fully transparent to the application developers, whilst delivering very high performance. For our experiments we have initialized OpenMPI such that we use the optical StarPlane links between the clusters, and the local Myrinet-10G network for intra-cluster communication.

To avoid confusing results, we have refrained from using multiple cores per CPU, or multiple CPUs per node. Also, single cluster runs always have been performed at the Vrije Universiteit (VU), using a maximum of 64 nodes (on DAS-2 and DAS-3, respectively). Dual-cluster runs always have been performed using the VU and Leiden (LU) clusters simultaneously, with an equal number of nodes on both clusters. Because it was not possible to use more than 24 nodes on either LU cluster, dual-cluster runs have been performed using a maximum of 48 nodes in total. Four-cluster runs have been performed in a similar manner, resulting in runs using a maximum of 96 nodes in total.

5.1 Performance and Speedup

From the description of the two versions of our application it is clear that the *ConvRot* version is more difficult to parallelize efficiently. This is due to the data dependencies in the applied algorithm (i.e., the repeated image rotations), and not due to the capabilities of Parallel-Horus. Hence, the *ConvRot* program is expected to have speedup characteristics that are not as good as those of the *Conv2D* program. However, *Conv2D* is expected to be the slower sequential implementation, due to the excessive accessing of image pixels in the 2-dimensional convolution operations.

Table 1, depicting the performance results for the two versions given a realistic orientation scale-space, shows that these expectations indeed are correct. On one node *ConvRot* is about 7 to 8 times faster than *Conv2D* on both DAS-2 and DAS-3. For 64 CPUs on a single cluster, and using our fast 'lazy parallelization' approach, this factor has dropped to 1.7 on DAS2, and 5.3 on DAS-3, respectively.

# CPUs	DAS-2: # Clusters			DAS-3: # Clusters		
	1	2	4	1	2	4
1	172.5	-	-	65.08	-	-
2	92.55	100.3	-	32.92	33.33	-
4	50.13	61.70	64.86	16.73	17.43	20.40
8	26.55	38.41	44.35	8.625	9.402	11.45
16	16.02	27.41	32.91	4.628	5.601	6.883
32	13.41	22.11	28.33	2.665	3.794	4.787
48	14.71	25.25	31.23	2.107	3.635	4.562
64	15.96	-	26.66	1.770	-	3.792
80	-	-	36.46	-	-	4.279
96	-	-	31.78	-	-	3.813

(a) ConvRot with Lazy Parallelization (results in seconds)

# CPUs	DAS-2: # Clusters			DAS-3: # Clusters		
	1	2	4	1	2	4
1	219.4	-	-	73.44	-	-
2	147.1	420.6	-	44.09	66.07	-
4	128.2	378.1	508.8	23.61	50.14	86.20
8	82.85	361.2	505.3	15.16	42.68	78.81
16	68.37	351.0	492.2	12.20	39.63	75.90
32	66.72	349.4	483.5	11.24	37.84	75.62
48	80.82	304.2	479.2	11.73	37.20	72.30
64	84.45	-	484.5	12.20	-	73.16
80	-	-	459.6	-	-	76.03
96	-	-	430.2	-	-	69.36

(b) ConvRot with Naive Parallelization (results in seconds)

# CPUs	DAS-2: # Clusters			DAS-3: # Clusters		
	1	2	4	1	2	4
1	1377.2	-	-	566.6	-	-
2	695.9	703.7	-	283.7	283.9	-
4	349.7	359.6	361.2	142.3	142.4	156.2
8	176.1	185.9	187.3	71.28	71.31	78.51
16	89.95	95.65	96.11	35.90	35.91	39.75
32	46.98	50.52	50.95	18.11	18.20	20.45
48	34.97	37.62	37.67	12.66	12.73	14.42
64	27.66	-	29.76	9.297	-	10.82
80	-	-	28.91	-	-	8.402
96	-	-	26.28	-	-	7.392

(c) Conv2D with Lazy Parallelization (results in seconds)

# CPUs	DAS-2: # Clusters			DAS-3: # Clusters		
	1	2	4	1	2	4
1	1433.4	-	-	572.9	-	-
2	748.1	954.8	-	290.2	366.0	-
4	409.3	616.4	713.2	150.9	169.9	209.7
8	241.1	435.6	524.7	76.74	97.71	129.9
16	152.7	344.5	460.2	41.59	44.07	89.67
32	111.4	299.3	446.7	27.00	43.94	70.29
48	92.84	243.3	347.0	20.10	36.93	64.34
64	87.86	-	369.0	17.19	-	61.66
80	-	-	349.4	-	-	60.71
96	-	-	372.6	-	-	56.04

(d) Conv2D with Naive Parallelization (results in seconds)

Table 1. Performance of ConvRot (a and b) and Conv2D (c and d) with 'lazy' and 'naive' parallelization. Results for computing an orientation scale-space at 5° angular resolution (i.e., 36 orientations) and 8 (σ_u, σ_v) combinations. Scales computed are $\sigma_u \in \{3, 5, 7\}$ and $\sigma_v \in \{1, 2, 3\}$, ignoring the isotropic case $\sigma_{u,v} = \{3, 3\}$. Image: 512×512 (4-byte) pixels.

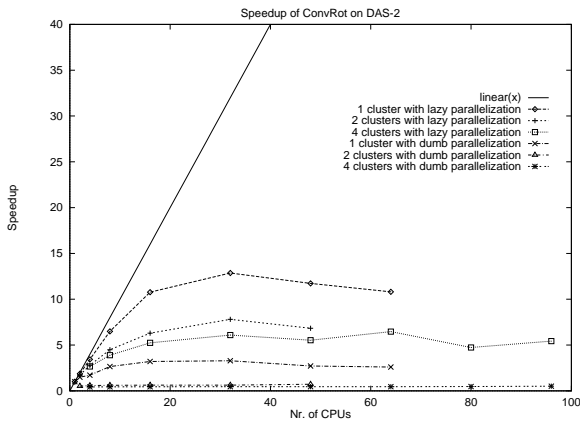


Figure 2. Speedup of ConvRot on DAS-2.

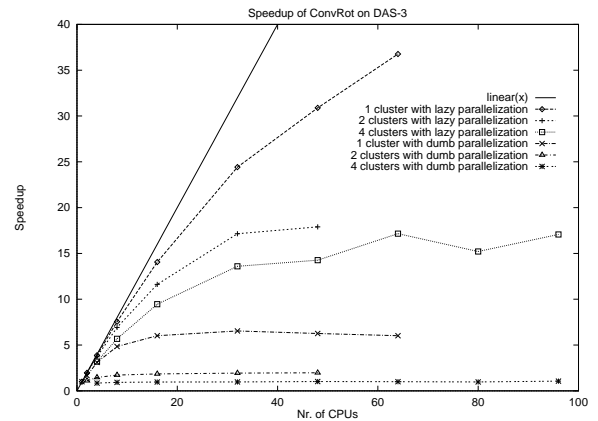


Figure 4. Speedup of ConvRot on DAS-3.

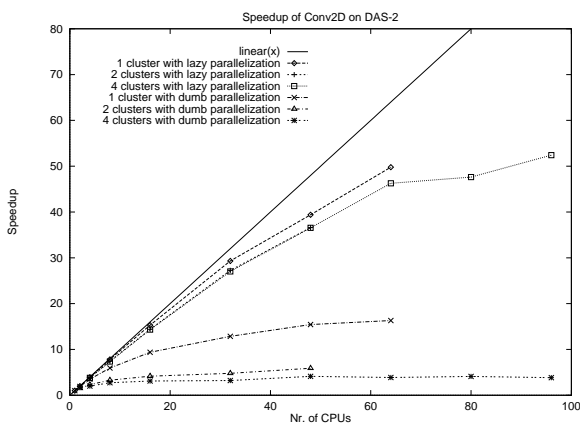


Figure 3. Speedup of Conv2D on DAS-2.

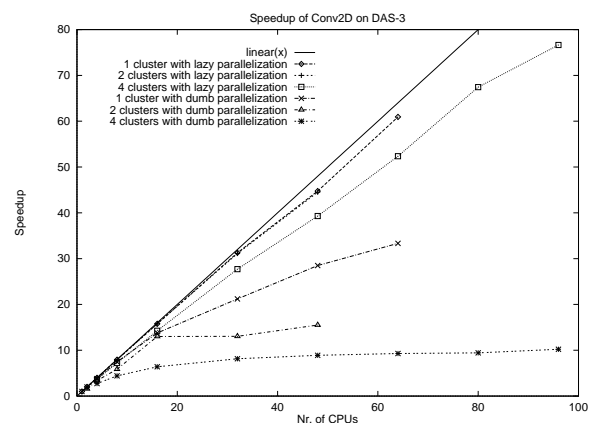


Figure 5. Speedup of Conv2D on DAS-3.

When comparing the multi-cluster runs in Table 1(a), it is clear that *ConvRot* indeed can benefit from an increased availability of nodes — in particular on DAS-3 — despite the fact that 64-node single-cluster runs always provide fastest execution. *Conv2D* shows the best multi-cluster results, with four-cluster results even outperforming the 64-node single-cluster runs.

From this performance comparison we conclude that, for all versions of our line detection problem, one can obtain increased performance from using nodes at multiple sites, in particular on DAS-3. As expected, a performance drop is introduced when moving from one cluster to multiple clusters, but such a move is still worthwhile in case the available number of nodes at one cluster is limited, or if the potential for parallelization of a problem at hand is (much) larger than can be exploited by a single cluster. In particular, for applications with good parallelization characteristics, such as our *Conv2D* program, the benefits can be significant.

Before we present our speedup analysis, we need to indicate that there is an apparent discrepancy between the 'lazy' and 'naive' single-node, single-cluster results. As

these runs essentially represent sequential execution in both cases, pairwise lazy/naive parallel runs on a single node of the same cluster should give identical performance results. This is not the case, however, as the presented results are obtained from parallel runs performed on a single node. In the lazy parallel case, the single-node parallelization overhead is such, that a factual sequential run can be said to be identical to a single-node lazy parallel run. In contrast, single-node naive parallel runs do show a significant performance drop, because of excessive creation, copying, and removal of internal parallelization structures.

When considering the speedup graphs of Figures 2-5, it is immediately apparent that DAS-3 provides much better speedup characteristics than DAS-2 — for single-cluster and multi-cluster runs alike. This is particularly remarkable given the fact that the sequential runs on DAS-3 are about 3 times faster than on DAS-2, thus reducing the theoretical potential for obtaining good speedups on DAS-3 relative to DAS-2. Our results are explained by the fact that the communication speeds relative to the computation speed have improved significantly on the new DAS-3 sys-

Program/Cluster	VU	LU	UvA	MN
ConvRot	65.08	60.66	70.95	65.65
Conv2D	566.6	523.4	617.7	566.7

Table 2. Single-cluster, single-node performance on each of the applied DAS-3 clusters.

tem. Importantly, this result indicates that our optical StarPlane network improves communication speeds over conventional Internet links by a similar (or even higher) relative amount as our local Myrinet-10G links improve communication speeds over the old Myrinet-2000 local interconnect.

One further significant result is found in Figure 5: the two-cluster lazy parallel runs provide speedup results that are *identical* to the single-cluster lazy parallel runs. Clearly, for applications with a good parallelization potential, multi-cluster runs constitute a realistic, and attractive alternative. In this respect one may wonder why the related four-cluster lazy parallel speedup results are somewhat lagging behind. This is explained by the fact that the four-cluster runs also incorporates the DAS-3 cluster at the University of Amsterdam, which has significantly slower compute nodes. As a result, in our current parallelization strategy (which results in *bulk synchronous* parallel execution) the faster compute nodes repeatedly have to wait for the slower UvA nodes. Table 2 shows the significance of the difference (i.e., even up to 18%) in the computation speeds among the different DAS-3 clusters applied in our experiments. Note that this effect, which is certainly not caused by any form of increased wide-area communication overhead, plays a role in all four-cluster results obtained on DAS-3.

Given the above results, we have clearly shown that — in contrast to common belief — fine-grained distributed data parallelism indeed can be a viable acceleration approach. Also, we conclude that DAS-3, with its optical interconnect, is a magnificent system for obtaining significant speedups, for single-cluster runs and multi-cluster runs alike.

6 Conclusions

In this paper we have applied a wide-area data parallelization approach to two different implementations of a well-known problem in multimedia content analysis. For the development of the applications we have used the Parallel-Horus multimedia computing library that allows programmers to implement data parallel applications as fully sequential programs. The applications have been tested on two different distributed systems, both consisting of multiple compute clusters located at different universities in The Netherlands. The DAS-3 system is equipped with a dedicated wide-area optical interconnect, called StarPlane, while DAS-2 uses conventional Internet links. We have presented experimental results obtained on both systems, and have provided a feasibility analysis of the applied parallelization approaches.

The most significant conclusion is that, while the compute nodes of DAS-3 are about 3 times faster than those of DAS-2, DAS-3 was still capable of providing much better speedup characteristics — even for larger numbers of CPUs, and for single-cluster and multi-cluster runs alike. This result is explained by the fact that the innovative local and wide-area networking technologies applied in DAS-3 (Myrinet-10G and StarPlane) provide speed improvements that are similar to, or even outweigh, the speed improvements obtained by the compute elements of DAS-3. In particular, given our results for the *Conv2D* version of our line detection program, we conclude that, for applications that have a good parallelization potential, multi-cluster runs constitute a realistic, and attractive approach. This result indicates that, in combination with the increasing adoption of optical interconnects in wide-area computing, there is a growing potential for scalable techniques for multimedia data analysis.

References

- [1] J. Geusebroek et al. A Minimum Cost Approach for Segmenting Networks of Lines. *Int. J. Comp. Vis.*, 43(2):99–111, July 2001.
- [2] R. Graham et al. Open MPI: A High-Performance, Heterogeneous MPI. In *Proceedings of HeteroPar'06*, Barcelona, Spain, Sep. 2006.
- [3] N. Karonis et al. MPICH-G2: A Grid-enabled Implementation of the Message Passing Interface. *J. Par. Dist. Comp.*, 63(5):551–563, May 2003.
- [4] D. Koelma et al. Horus C++ Reference. Technical report, Univ. Amsterdam, The Netherlands, Jan. 2002.
- [5] J. Lebak et al. Parallel VSIP++: An Open Standard Software Library for High-Performance Signal Processing. *Proc. IEEE*, 93(2):313–330, Feb. 2005.
- [6] H. Mohamed and D. Epema. Experiences with the KOALA Co-Allocating Scheduler in Multiclusters. In *CCGrid2005*, Cardiff, UK, May 2005.
- [7] P. Morrow et al. Efficient Implementation of a Portable Parallel Programming Model for Image Processing. *Concur.: Pract. Exp.*, 11:671–685, Sep. 1999.
- [8] A. Plaza et al. Commodity Cluster-based Parallel Processing of Hyperspectral Imagery. *J. Par. Dist. Comp.*, 66(3):345–358, Mar. 2006.
- [9] F. Seinstra et al. Finite State Machine-Based Optimization of Data Parallel Regular Domain Problems Applied in Low-Level Image Processing. *IEEE Trans. Par. Dist. Syst.*, 15(10):865–877, Oct 2004.
- [10] F. Seinstra et al. User Transparency: A Fully Sequential Programming Model for Efficient Data Parallel Image Processing. *Concur. Comp.: Pract. Exp.*, 16(6):611–644, May 2004.
- [11] F. Seinstra et al. High-Performance Distributed Video Content Analysis with Parallel-Horus. *IEEE Multimedia*, 14(4):64–75, Oct.-Dec. 2007.
- [12] C. Snoek et al. The Semantic Pathfinder: Using an Authoring Metaphor for Generic Multimedia Indexing. *IEEE Trans. Pat. Anal. Mach. Intel.*, 28(10):1678–89, Oct. 2006.