Noname manuscript No.
(will be inserted by the editor)

Scalable shot boundary detection

Pablo Toharia · Oscar D. Robles · Jose L. Bosque · Angel Rodríguez

Abstract In a Content-based Video Retrieval system the shot boundary detection is an unavoidable stage. Such a high demanding task needs a deep study from a computational point of view to allow finding suitable optimization strategies. This paper presents different strategies implemented on both a shared-memory symmetric multiprocessor and a Beowulf cluster, and the evaluation of two different programing paradigms: shared-memory and message passing. Several approaches for video segmentation as well as data access are tested in the experiments, that also consider load balancing issues.

Keywords
Shot boundary detection - Information Storage and Retrieval - Distributed systems - Performance evaluation - Load balancing

1 Introduction

Content-based Multimedia Retrieval (CBMR) systems provide efficient access in order to allow users querying multimedia databases so they can retrieve the most similar items from the datasets [6,8].

In the case of video data, the unavoidable first stage is Shot Boundary Detection (SBD), that deals with the extraction of the minimum units with semantic meaning: shots. This way, this type of algorithms, besides isolating

Pablo Toharia · Oscar D. Robles
Dpto. de Arquitectura y Tecnología de Computadores y Ciencia de la Computación e Inteligencia Artificial
Universidad Rey Juan Carlos E-mail: {pablo.toharia,oscardavid.robles}@urjc.es
Jose L. Bosque
Dpto. de Electrónica y Computadores
Universidad de Cantabria. E-mail: joseluis.bosque@unican.es
Angel Rodríguez
Dpto. de Arquitectura y Tecnología de Sistemas Informáticos
Universidad Politécnica de Madrid E-mail: arodri@fi.upm.es

Table 1 Execution time and maximum speedup under Amdahl's law (S_∞) for different Zernike polynomials (time values are expressed in seconds).

Method	Video size (frames)	Dec. time	Seg. time	Total time	Serial frac. (%)	Parallel frac. (%)	S_∞
zer3	5000	62.80	293.20	356	17.64	82.36	5.67
	10000	125.84	586.16	712	17.67	82.33	5.66
	20000	252.34	1172.66	1425	17.71	82.29	5.65
zer5	5000	62.88	587.12	650	9.67	90.33	10.34
	10000	125.70	1174.30	1300	9.67	90.33	10.34
	20000	252.74	2350.26	2603	9.71	90.29	10.30
zer10	5000	63.00	2124.00	2187	2.88	97.12	34.71
	10000	125.84	4249.16	4375	2.88	97.12	34.77
	20000	254.11	8503.89	8758	2.90	97.10	34.47

shots try to make it easier to process video contents more efficiently. Depending on the video signal, these algorithms work on a compressed [1] or a non-compressed [11] domain. This paper is focused on non-compressed video segmentation because the primitives used are also part of the retrieval stage in a Content-based Video Retrieval system [7].

Such a high demanding process, as SBD is, needs efficient and scalable implementations that allow completing this stage achieving low response times, taking into account that the size of the data to be processed keeps growing day by day. Then, this paper firstly begins with a deep and exhaustive analysis of a typical SBD algorithm, that includes experimental tests over suitable architectures, adequate paradigms and efficient implementations. The results allow to extract relevant conclusions about the set of parameters that achieve the best results for the most suitable architecture from the point of view of both performance and scalability. Thus, the performance of the SBD algorithm is tested on two implementations based on two different programming paradigms: shared-memory communication and distributed memory processing using message passing paradigm. Considering these software solutions, a shared-memory symmetric multiprocessor (SMP) and a distributed system have been chosen to compare both implementations. It can be said that the use of distributed solutions on clusters comes from their remarkable scalability, flexibility and fault tolerance [3,2], that makes them providing an interesting cost/performance ratio to solve this problem.

Shot boundary detection algorithms basically compute differences between consecutive frames or groups of frames moving a symmetric mask window over the whole set of frames. These differences can be computed based on different visual features [5,11], although in this case, due to previous experience, Zernike invariants [10], have been used. The algorithm follows the model of Zhang *et al.* [12]; low-level details can be found in [7]. One key aspect is that video data, coming from the TRECVid project [9], is coded using MPEG. This means the whole process has to be broken down in two stages: a first one to decode the video and a second one that accomplishes the video shot extraction on noncompressed frames.

Table 1 shows decoding and segmentation times obtained for different videos. Labels `zer3`, `zer5` and `zer10` stand for Zernike invariants up to 3^{rd}, 5^{th} and 10^{th} order respectively. It can be seen that the higher the order is,

the bigger the execution time of the segmentation stage is, reaching almost 2 hours and a half for videos of 20,000 frames in the case of `zer10`. It can be also pointed out that the effort of looking for efficient and scalable implementations must be done on the SBD, since the great computational load is on this stage rather than on the decoding one. Amdahl and Gustafson laws [4] say that the bigger is the size of the problem to solve, the greater is the maximum achievable speedup if the serial fraction of the program does not grow with the problem size. Also, the serial fraction of the different parallel solutions available overlap in some way with the parallel fraction of the problem, i.e. the segmentation stage. As the number of nodes in the system grows, so does this overlapping. This is the reason why the system size also influences the maximum achievable speedup, which is also affected by the problem size.

Table 1 shows that the parallel fraction grows with the polynomial degree, while the serial fraction remains constant. At the same time, the decoding and segmentation times linearly grow with the video size. Since there are dependencies among the different video frames, the decoding stage must be solved sequentially. But the values in the table show the benefits of using a parallel implementation, specially when using high order polynomials.

Upon observing the operations involved in the extraction stage, it can be deduced that all processed frames have data dependencies, but only those belonging to shot boundaries are critical to compute the exact points where shots begin and end. It means that an approach based on data decomposition could be achievable if the boundary and the frames before and after it are sent to the same thread or process. Therefore, the SBD stage could be fully parallelized.

2 Approaches and strategies implemented

As mentioned before, this paper deals with two different parallel architectures (**SMP** and **Cluster**) and with two different parallel programming paradigms. The first solution proposed is based on the shared-memory paradigm. It has been implemented using LinuxThreads because Posix libraries are more efficient than OpenMP. This solution is only feasible to be implemented on the SMP architecture. The second solution has been programmed using MPI libraries as communication primitives between *master* and *slave* processes. The main advantage of these implementations is that they can also be used on SMP architectures, so they have been tested on both architectures.

Both distributed and centralized decoding approaches have been developed. The differences between them lay on the decoding stage, as well as on the way data are accessed. Considering the cluster architecture, since input data are stored in one of the nodes, a farm based structure has been selected, in which the *master* distributes the workload among the *slave* processes and collects the partial results processed in each *slave* to obtain the video shots. In this case two alternatives are also proposed: static and dynamic data distribution. These strategies can be tested also in the SMP computer.

Distributed decoding approach (DDA). Each thread has to access and decode video data and to perform shot detection tasks, so there is not any *master* thread. Prior to compute the shot detection algorithm, each thread has to find its assigned starting location. The algorithm is based on the computation of an adaptive threshold computed over a sliding window [7], so a few frames before the assigned beginning location and after the ending location must be considered. Due to this, there will be overlapping, but no dependencies, among consecutive slices. The output of each thread will be an ordered list of detected cuts including the exact first and last frame numbers of each shot.

Centralized decoding approach (CDA). In this approach a *master* thread[1] decodes chunks of video data and passes them to the *worker* threads. Both decoding and segmentation stages are processed independently, mainly because the decoding time is much lower than the segmentation time. This solution avoids the bottleneck that appears when multiple threads are performing simultaneous positioning over the same video data, as it can happens with **DDA**.

Static data distribution (SDD). The *master* process does an initial and homogeneous data distribution among the *slave* processes. The size of data packages is obtained dividing the total number of frames in the video by the number of available processors. In this approach, the *master* begins a decoding loop and sends a complete data package to each *slave*. *Slave* processes send the results back to the *master* after finishing the segmentation stage. The *master* gathers these partial results and stores them.

Dynamic balanced data distribution (DDD). When the available nodes in the system, because of a hardware heterogeneity or a non-dedicated use, a serious workload imbalance can appear. To prevent it, this approach implements a dynamic, global and centralized load balancing algorithm. The video is divided into a certain number of equally sized work packages, greater than the number of available nodes in the system. The *master* sends one work package to each available node and waits for the answers. As soon as *slaves* finish their assigned work they ask for new work packages, that the *master* will send while there are pending ones. This approach allows the most powerful nodes to process a higher number of work packages, obtaining a quite similar response time for all nodes, regardless of their hardware and current workload. This advantage has a more remarkable impact in very dynamic systems where the local workload of the nodes can vary drastically along the execution of the application.

DDD implementation has greater communication overhead than **SDD** approach, since more messages are exchanged. However, it is a quite reduced increase due to the fact that because messages are quite big, the amount of information they carry is always the same, and communication is limited by the available bandwidth on top of other factors. For all these reasons, it would be an optimum solution to choose during the execution between **SDD** and **DDD** approaches based on both system's dynamism and heterogeneity.

[1] From now on there will be no difference on the use of the terms thread and process.

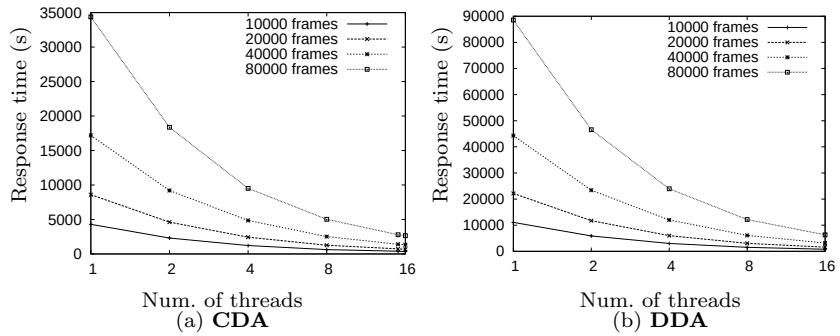


Fig. 1 **DDD** response time with LinuxThreads on **SMP**: **CDA** vs. **DDA**.

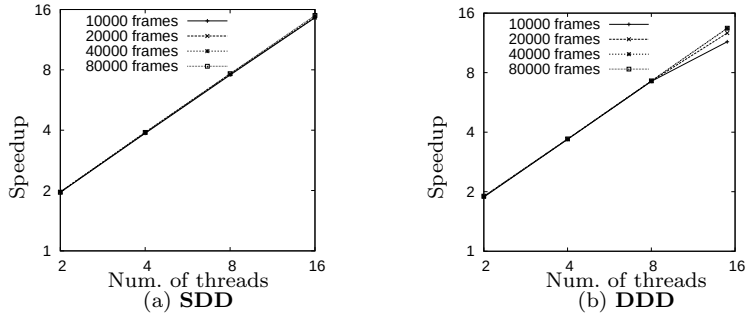


Fig. 2 Speedup obtained with LinuxThreads on **SMP**: **SDD** vs. **DDD**.

3 Experimental results

The experiments have been performed on a shared-memory CC-NUMA based symmetric multiprocessor (**SMP**), a SGI Prism 350 machine with 16 Intel Itanium II processors and a main memory of 32GB DDRAM, and on a cluster setup (**Cluster**) made up of eServer BladeCenter JS20 (2 IBM Power 970 2.2 GHZ processors and 4 GB of main memory) nodes, where due to administration issues up to 64 of them were available. Along the experiments, the optimized video segmentation has been run for different videos, number of threads and number of cluster nodes. It must be noticed that in **CDA** strategy each *slave* process is assigned to one of the processors, plus one processor dedicated to run the *master* code, with the exception of the setup with 16 nodes, which runs one *master* and one *slave* in one node, and 15 *slave* processes in the other nodes. The main goals of the experiments are: (1) to validate the viability of a parallel solution for the video segmentation application on different architectures and parallel programming paradigms; (2) to compare the performance of two alternative architectures, based on shared-memory and on distributed memory; (3) to compare two parallel computation paradigms, like shared-memory *vs.* message passing programming and different strategies based on data access and data processing; (4) to test dynamic data distribution strategy with load balancing mechanisms. **CDA** can only be compared against **DDA** in the **SMP**, since **DDA** structure does not fit well in the **Cluster**. **SDD** and **DDD** strategies have been compared in both architectures.

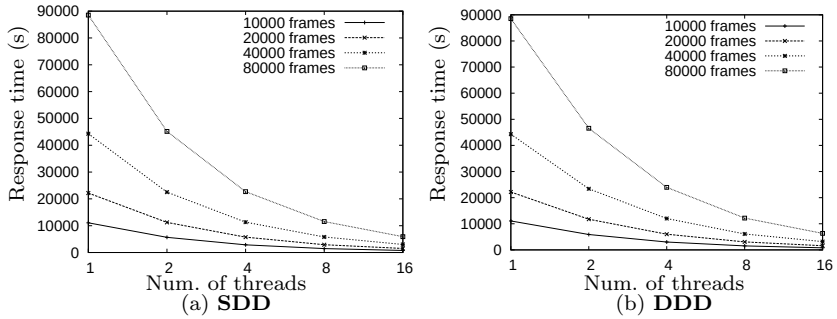


Fig. 3 **DDA** Response time with LinuxThreads on **SMP**: **SDD** vs. **DDD**.

3.1 Comparison among different strategies

*3.1.1 Comparison between **CDA** and **DDA** in the **SMP***

Figure 1 presents the evolution of response times and Fig. 2 the speedup[2] for both implementations when the number of processors is increased and several data sizes are considered. To obtain these results, packages of 157 frames have been used. With this size, the different processors involved have to exchange different communication operations. In these cases, figures obtained improve the ones presented as both video and package sizes are increased. All figures show a great reduction of the response times, so it can be deduced that parallel implementations are a good solution to cut down the application response time.

A detailed analysis of the results presented in Fig. 1 and Fig. 2 reveals that the **CDA** response times are always at least twice better than **DDA** ones. It can be noticed how speedup follows a nearly linear curve as a function of the number of considered processors. Moreover it must be said that speedup is almost independent of the input data size growth, so it can be concluded that the communication overhead is negligible with regard to the processing time.

*3.1.2 Comparison between **SDD** and **DDD***

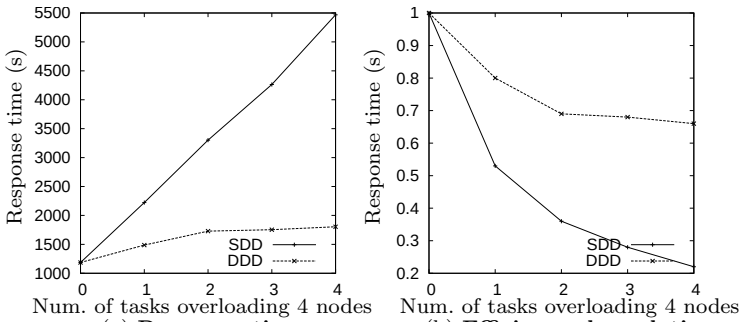
The performance achieved by both solutions has been tested in both architectures.

SMP: Figure 3 presents the evolution of the response time for both implementations in **SMP**. The package size for **DDD** is again fixed to 157 frames, while in **SDD** the greatest package size is fixed by the test with the maximum number of nodes (15 in **DDD**). All figures show a great reduction of response times, with **SDD** version slightly outperforming **DDD** approach. It can be said that the speedups are very close to the ideal one. The execution time decreases while the package size grows, except for the setup with 15 processors. Anyway, the **SDD** version still beats **DDD** with its maximum package size. The worse results in **DDD** with 15 processing nodes are due to the distribution of the workload, that involves processing the last package in only one node while the rest are idle, a fact that increases the total execution time.

[2] The speedup for each implementation is calculated as the ratio between the time obtained when using a single processor and the time with N processors.

Table 2 Task load distribution when **DDD** introduces load balancing mechanisms.

Initial load (# of tasks)	Processor #			
	[1-4]	[5-8]	[9-12]	[13-16]
0	8	8	8	8
1	5	9	9	9
2	4	10	9	9
3	3	10	10	9
4	2	10	10	10



(a) **Response time** (b) **Efficiency degradation**

Fig. 4 DDA response time and efficiency degradation of the LinuxThreads version on **SMP** with load balancing: **SDD** vs. **DDD**.

Some of the processors have been overloaded in order to run load balancing tests on **SMP**, although the results obtained can be extrapolated since this study is architecture independent. The performance of the system dealing with 20,000 frames and 157 frames per package is presented. Table 2 shows the number of packages processed by each node of the **SMP** architecture when 4 of these processors are executing a limited number of additional tasks, ranging from 0 (no overload at all) to 4 (maximum overload). As it is shown in these values, the workload is distributed among the unloaded processors (from node 5 to 16) depending on the number of packages to distribute from an initial workload of 128 packages. Figure 4 shows the response time and efficiency degradation achieved by **SDD** and **DDD** overloading four nodes in **SMP** as described in Table 2. As it can be noticed in both figures, load balancing improves the performance of the whole system offering good levels of efficiency, although there are several nodes dedicated to attend other jobs. However, in a homogeneous system without any external workload, **SDD** will achieve better performance because of the absence of load balancing overhead. As mentioned above, these experiments can be generalized to other architectures like homogeneous *clusters* with imbalanced nodes or heterogeneous *clusters*.

Cluster: In these case and due to their interest, the speedups achieved by both implementations in **Cluster** are shown in Fig. 5. In **DDD**, the increment of the speedup as the video size grows can be considered spectacular with the one obtained for **SDD**, reaching the minimum with 10,000 frames, surpassing 50.6 with 20,000 frames (with an efficiency value of 0.8) and reaching a maximum value of 57.5 with 80,000 frames (with an efficiency value of 0.9).

This study measures communication times on both implementations. Figure 6 shows the percentage of total execution time devoted to communication

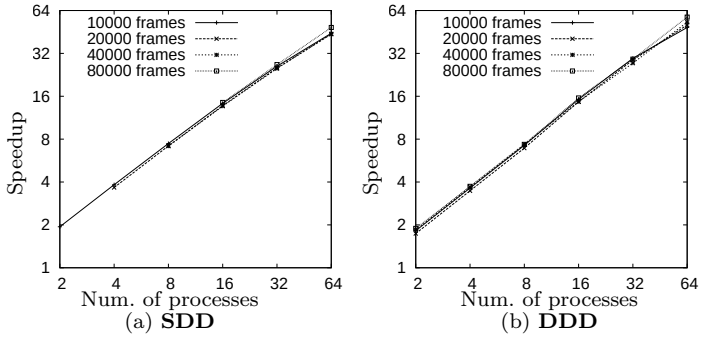


Fig. 5 CDA Speedup of the **MPI** versions on **Cluster** comparing **SDD** and **DDD**.

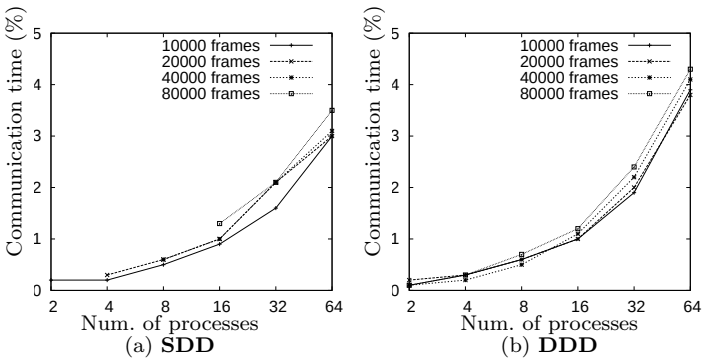
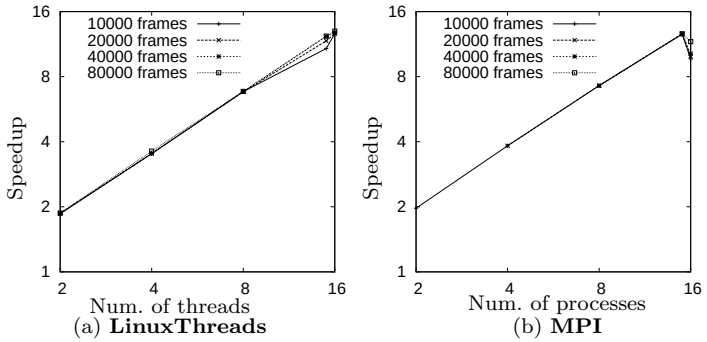


Fig. 6 CDA Response time percentage dedicated to communication operations of the **MPI** versions on **Cluster** comparing **SDD** and **DDD**.

data. The communication time only depends on the video size, remaining almost constant for all the setups tested in each size. On the other hand, it can be seen that the behavior is very similar for all sizes tested. Finally, it must be noticed that the increment of the communication time does not cause the percentage increase shown in Figure 6. This increase is due to a reduction on the processor workload when more processors are involved in the setup.

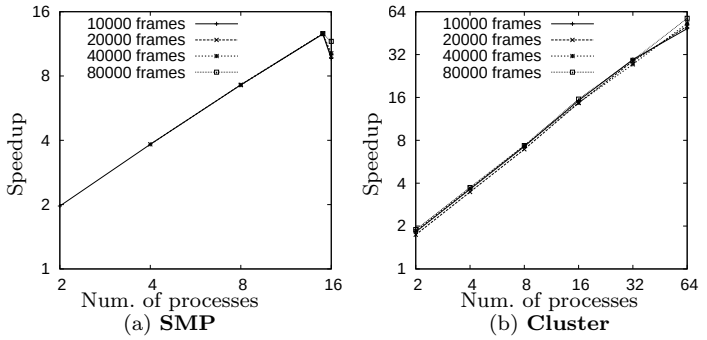
3.2 Comparison of shared-memory vs. message passing

This section presents the results comparing both programming paradigms, shared-memory and message passing. Figure 7 presents the evolution of the speedup for both programming paradigms in **SMP**. All figures show quite outstanding speedup curves. When the video size is greater than 20,000 frames, **LinuxThreads** version improves the results provided by **MPI**. Figure 7(b) clearly shows the degradation that appears when the processor dedicated to run the *master* process in the **MPI** version is shared by an additional 16^{th} *slave* process, overloading the corresponding **SMP** processor. This means that the load associated to the *master* task is not negligible with **MPI**, while sharing one processor by the main thread of the **LinuxThreads** and one of the launched threads does not diminish speedup figures, as Figure 7(a) shows.



(a) **LinuxThreads** (b) **MPI**

Fig. 7 CDA-DDD Speedup on **SMP** comparing implementations of shared-memory (**LinuxThreads**) and message-passing (**MPI**) paradigms.



(a) **SMP** (b) **Cluster**

Fig. 8 Speedup on both architectures using the same programming paradigm (**MPI**) and the same strategy for distributing the load among the available processors (**CDA**).

3.3 Comparison between parallel architectures: SMP vs. cluster

Finally, this section presents a comparison between both architectures using the same programming paradigm and the same implemented approach. Better response times in **Cluster** can be achieved since the number of available processors is greater than in **SMP**, but both architectures can be compared using speedup figures, which are presented in Fig. 8. Again, all figures show a great reduction of response times and quite outstanding speedup curves. When the number of *slaves* in the cluster is increased, the slope increment of the curve is clearly sharper in **Cluster** than in **SMP**, proving a better scalability in the **Cluster** than in **SMP**. The degradation that appears in Fig. 8(a) is the same one shown in Fig. 7(b) for previously commented reasons. It must be noticed that to maintain the speedup and then the scalability, one processor cannot share both *master* and *slave* processes, so it is necessary to leave exclusively one processor for the *master*, as seen in Figures 7(b) and 8(a) when using 15 *slaves*.

4 Conclusions and future work

This paper presents a quite exhaustive study among different parallel architectures and parallel programming paradigms of a video shot segmentation algorithm used in a Content-based Multimedia Retrieval System. Programmed implementations attempt to cover all possible parallel programming aspects,

just as the different studied paradigms: two LinuxThreads implementations and two MPI versions, tested on a shared-memory symmetric multiprocessor and on a cluster. Important conclusions can be extracted from these experiments. Considering the strategy for accessing and segmenting video data, it can be stated that **DDA** has a much simpler implementation, although **CDA** is clearly better from the point of view of performance and shows a slightly better scalability. Simultaneous access of the threads to the same video data for decoding purposes means a bottleneck in the I/O subsystem that surpass the penalty introduced when only a single thread is devoted to do the task.

The comparison of both distribution strategies, **SDD** vs. **DDD**, when there are no problems with memory accesses, turns out that differences are almost negligible and the simplest one should be chosen (**SDD**). But when video size grows or when dealing with heterogeneous clusters, **DDD** approach is better to introduce load balancing solutions as it has been experimentally shown.

The performance obtained by both programming paradigms is very similar in **SMP** with both implementations. Programming LinuxThreads is simpler than MPI, but message passing enjoys portability as an unquestionable advantage over threads, since the same code can be executed in both architectures.

From the experiments it can be deduced that all implementations and architectures tested achieve excellent performance, with strong reductions of execution times. Speedup values are almost linear and quite close to the theoretical maxima in all experiments. Shared memory architectures obtain better results with a small number of processors because each one is more powerful than the nodes available in the cluster, but are less scalable than clusters.

Beowulf clusters with quite powerful networks achieve excellent scalability results. Communication times remain almost constant when the number of nodes grows, then the speedup achieved by the parallel system keeps stable when the problem size grows and the number of nodes also increases.

Future work will include the integration of these systems in a grid infrastructure. Another issue to be considered will be the implementation of the scalable SBD system in other available many-core architectures, such GPUs.

Acknowledgements

This work has been partially funded by the the Spanish Ministry of Education and Science (grants TIN2010-21289, TIN2010-21291-C02-02, Consolider CSD2007-00050 and Cajal Blue Brain project).

References

1. Sameer Antani, Rangachar Kasturi, and Ramesh Jain. A survey on the use of pattern recognition methods for abstraction, indexing and retrieval of images and video. *Pattern Recognition*, 35(4):945–965, April 2002.
2. Gordon Bell and Jim Gray. What's next in high-performance computing? *Communications of the ACM*, 45(2):91–95, February 2002.
3. José Luis Bosque, Óscar D. Robles, Luis Pastor, and Angel Rodríguez. Parallel CBIR implementations with load balancing algorithms. *Journal of Parallel and Distributed Computing*, 66(8):1062–1075, August 2006.
4. John L. Gustafson. Reevaluating Amdahl's law. *Commun. ACM*, 31(5):532–533, 1988.

5. Robert A. Joyce and Bede Liu. Temporal segmentation of video using frame and histogram space. *IEEE Transactions on Multimedia*, 8(1):130–140, February 2006.
6. Oge Marques and Borivoje Furht. *Content-based Image and Video Retrieval*. Multimedia Systems and Application Series. Kluwer Academic Publishers, USA, 2002.
7. Óscar D. Robles, Pablo Toharia, Angel Rodríguez, and Luis Pastor. Towards a content-based video retrieval system using wavelet-based signatures. In M. H. Hamza, editor, *7th IASTED International Conference on Computer Graphics and Imaging - CGIM 2004*, pages 344–349, Kauai, Hawaii, EEUU, August 2004. IASTED, ACTA Press.
8. Nicu Sebe, Michael S. Lew, and Arnold W. M. Smeulders. Video retrieval and summarization. *Computer Vision and Image Understanding*, 92(2–3):141–146, 2003.
9. Alan F. Smeaton. TRECVID 2003 video evaluation overview. Presented at the TRECVID 2003 Conference, National Institute for Standards and Technology, November 2003. `http://www-nlpir.nist.gov/projects/tvpubs/tvpapers03/tv3.overview.slides.pdf`.
10. Pablo Toharia, Óscar D. Robles, Ángel Rodríguez, and Luis Pastor. A study of Zernike invariants for content-based image retrieval. In *Proceedings of the IEEE Pacific Rim Symposium on Image Video and Technology, PSIVT2007*, volume 4872 of *Lecture Notes on Computer Science*, pages 944–957, Chile, December 2007. IEEE, Springer Verlag.
11. G. Valencia, J. A. Rodríguez, C. Urdiales, and F. Sandoval. Color-based video segmentation using interlinked irregular pyramids. *Pattern Recognition*, 37(2):377–380, February 2004.
12. Dong Zhang, Wei Qi, and Hong Jiang Zhang. A new shot boundary detection algorithm. In Heung-Yeung Shum, Mark Liao, and Shih-Fu Chang, editors, *IEEE Pacific Rim Conference on Multimedia*, volume 2195, pages 63–70. IEEE, Springer, October 2001.