Combining Flash Memory and FPGAs to Efficiently Implement a Massively Parallel Algorithm for Content-Based Image Retrieval

Rayan Chikhi[1], Steven Derrien[2], Auguste Noumsi[3], and Patrice Quinton[4]

[1] ENS Cachan, antenne de Bretagne – Bruz Cedex, France
[2] IRISA/Université de Rennes 1 – 35042 Rennes Cedex, France
[3] IRISA/Université de Douala – Daouala, Cameroun
[4] IRISA/ENS Cachan, antenne de Bretagne – Bruz Cedex, France

Abstract. With ever larger and more affordable storage capabilities, individuals and companies can now collect huge amounts of multimedia data, especially images. Searching such databases is still an open problem, known as content-based image retrieval (CBIR). In this paper, we present a hardware architecture based on FPGAs which aims at speeding-up visual CBIR.Our architecture is based on the unique combination of reconfigurable resources combined to Flash memory, and allows for a speed-up of 45 as compared to existing software solutions.

1 Introduction

With large storage devices becoming more and more affordable, individuals and companies can collect huge amounts of information, e.g. multimedia data. Many activities such as journalism, medical diagnosis or crime prevention rely on large multimedia (mostly images) databases. To use such databases efficiently, users must be able to browse and query their images according to their *content*. These types of search operations are usually referred to as Content-Based Image Retrieval (CBIR) and they are a very active research area [9,1,3].

There are mainly two types of CBIR methods. The first one is based on the semantic content of the image. It consists in finding images in a database that match a user query (e.g. keywords) which describes the semantic content of the image sought. Such an approach requires each image to be annotated with its semantic content. This annotation can be either done manually or automatically. Most approaches for automated annotation usually rely on textual data which is associated to the image, for example, text in the enclosing web page.

The second method is based on the visual content of the image. It relies on complex image processing algorithms which extract *image descriptors* summarizing the image visual content. A typical use of this approach is digital content copyright enforcement, which is a big concern for image database copyright owners such as photo agencies. In this case, the goal of the copyright owner is to retrieve – typically from the web – all unregistered uses of its images. In such a context, the retrieval technique must be very robust to image transformations, as the original image might have undergone several transformations such as cropping, compression, color change, etc.

In this work we focus on visual content based image retrieval methods. These methods share characteristics which make them very interesting candidates for hardware acceleration :

- They suffer from prohibitive execution time. For example, searching a 30,000 image database requires 15 minutes of execution time on a standard PC workstation.
- They are computationally intensive since they rely on euclidian (i.e. L_2) distance calculation in higher dimension vectors.
- They involve very large databases: typical image databases range from a few thousands to tens of millions of images.

We propose an application-specific parallel architecture for speeding-up visual CBIR methods. This architecture is designed as a target application for the ReMIX machine, a reconfigurable accelerator aiming at content processing for large databases. The specificity of the ReMIX machine is its unique combination of high throughput, very large storage resource based on Flash technology, and of high performance FPGA technology in order to speed-up search problems.

The remaining of this paper is organized as follows. Section 2 provides background information regarding the type of CBIR algorithms we are interested in. Section 3 presents the ReMIX platform, both at the system and at the architectural level. Section 4 presents our hardware implementation strategy. Results are given and discussed in Section 5. Conclusion and future work directions are sketched in Section 6.

2 Content-Based Image Algorithms

2.1 A Short Introduction to Visual CBIR Methods

Visual Content-Based Image Retrieval consists in searching an image database to retrieve images which are visually similar to a given query image. This type of operation is based on the notion of *image descriptor*. An image descriptor can be seen as a *signature* which is computed directly from visual features of the image such as colors, shapes, contrast, etc.

Two types of descriptors can be used: *global descriptors* encode an overall property of the image such as its color histogram, while *local descriptors* are only related to specific points of interest in the image (see Fig. 1). Using such *descriptors* simplifies the use of image databases, since it allows the user to search among the descriptor database rather than the whole image database.

In this paper, we are interested in *local descriptors*, since it has been shown that they provide a more robust approach to CBIR.

2.2 Extracting Local Descriptors

Retrieving an image consists first in computing the set of descriptors of the reference image – typically, a few hundred vectors of n real components. These descriptors are extracted from small regions of the image that contains specific visual features (these

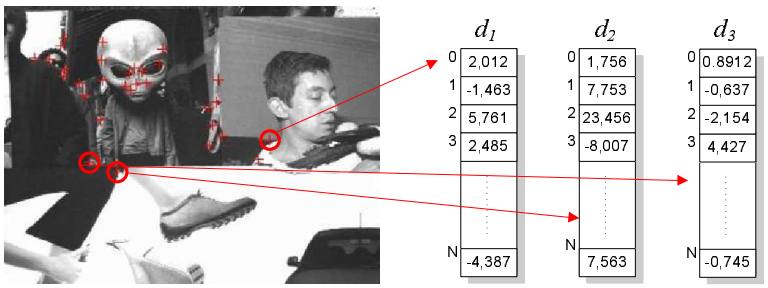

Fig. 1. Point of interest in an image and their associated local descriptors

regions are called interest points), using complex image processing algorithms (see for example Mikolajczyk et al. [10] for an overview of these techniques).

In addition to offering a concise description of the image content, these descriptors also need to be *robust* to image transformations. In other words, descriptors should be able to identify an image, even though it underwent several image transformations, such as cropping, constrast/light change, rotation, compression, etc. In our case an image is typically represented by a set of 50 to 1500 descriptors, each one being a 24-dimension real vector.

2.3 CBIR With Local Descriptors

Once extracted, the reference image descriptors (q_i) are compared to the image database descriptors (b_i) according to the metric $d(b_j, q_i)$ which corresponds to the euclidian distance (*distance calculation stage*):

$$d(b_j, q_i) \triangleq \sum_{n=0}^{24} |b_{j,n} - q_{i,n}| \quad .$$

For each reference descriptor q_i, a k-NN sorting (k-NN stands for k-nearest neighbors) selects the k database descriptors, the distances $d(b_j, q_i)$ of which are the smallest (*selection stage*). Finally, votes are assigned to images depending on their occurrences in the k-nearest neighbor lists (*election stage*): the image that has the largest number of votes is considered the best match.

As mentioned in the introduction, retrieving an image in a 30,000 image database requires about 15 minutes on a standard workstation. This is impractical for most applications of CBIR, since they often require a low response time. Research on smarter algorithms, based on clustering techniques for example, although very active, has not lead to definitive results because of a phenomenon called *curse of dimensions* which affects large databases operating on higher-dimensional data sets [1,3]. Very recently, search methods based on list-ranking have been proposed [9]. Although they offer approximate results, these methods happen to be very efficient in terms of response time. For example, using this approach, searching a 20,000 image database takes less than twenty seconds.

It is therefore questionable whether there is any interest in speeding-up the original sequential scan which is based on exhaustive search. In practice, the sequential scan search is of great use for the community that studies descriptors and search algorithms. When introducing a new type of descriptor extraction or encoding, there is a need for validating its efficiency and robustness. To obtain unbiased results, researchers need to benchmark their descriptors by using large databases which must be scanned completely to obtain exhaustive results. This is usually a very time consuming process since for each image of the database, a large number (a few hundred) of image variants are generated. Each variant is to be matched against the whole database, and this operation is repeated for a significant subset of the database. This represents a huge volume of computation (in the order of weeks or months) since the larger the database is, the more valuable the search results are.

2.4 Related Work

Accelerating CBIR on a parallel machine is the most natural choice, and has already been studied by a few authors, among whom Robles et al [14]. There has been only some work on special-purpose hardware for this type of application [7,15,11]. However, most of this work either did not address the problem in the context of a *real-life* hardware system (i.e. with its communication interface, I/O bandwidth constraints, etc.), or considered a very naive algorithm which has no interest in practice. In a previous work [12], we proposed to accelerate the CBIR algorithm on a FPGA based smart-disk architecture [5]. While this approach provided interesting performance improvement, its efficiently was greatly affected by the limited sustained hard-disk throughput. While modern hard-drive interface such as SATA offer data transfer bandwidth up to 133 MBps, the hard disk internal I/O rate when performing sequential scan is much lower. To overcome this difficulty, we propose an improved architecture which can take advantage of a very high data throughput, by handling both *distance computation* and *selection* in hardware.

3 The ReMIX Platform

3.1 Overview

The ReMIX machine is a reconfigurable accelerator targeted at content processing for very large unstructured and indexed databases. The idea behind ReMIX is to benefit simultaneously from the very high data throughput and the short access-time that can be obtained with the parallel use of Flash memory devices on the one hand, and from the high computing density of a high-end FPGA on the other hand. As such, this principle follows the philosophy of *intelligent memory* proposed by Patterson et al. in the context of the IRAM project [13].

The goal of the ReMIX architecture was to design a reconfigurable accelerator that could easily be integrated within a host system and would have the largest possible storage capability and the smallest possible random access time. We considered SRAM,

DRAM, magnetic storage and FLASH memory as possible candidate storage technologies. Table 1 summarizes the characteristics of these technologies with respect to density, cost, throughput and random access time in late 2006. All numbers shown assume a 64 gigabyte memory with a 64 bit width data bus.

Technology	# of chips for 64 GB	Cost	Access Time	Bandwidth	Total power
SRAM	7280	$ 123,500	5 ns	800 MBps	5250 W
SDRAM	512	$ 4,115	10 ns	2 GBps	30 W
Flash -NAND	64	$ 1,030	25 us	160 MBps	450 mW
Flash -NOR	4096	$ 72,500	100 ns	320 MBps	550 mW

Table 1. Memory and storage technology in late 2006

These figures tell us that SRAM technology is obviously not suited to build large size memory systems. SDRAM could be a good candidate; However integrating 64 GB of SDRAM memory on a single PCB device remains a very challenging problem because of power signal integrity issues. On the other hand, NAND-Flash technology is probably the best solution: it offers storage densities above those of DRAM (in late 2006 32 Gb NAND-Flash are available, while only 1 Gb for SDRAM), and this gap is expected to grow in favor of Flash memory in the forthcoming years. Nevertheless, NAND-Flash exhibits significant differences with standard memories in the way data is accessed. In NAND-Flash data is addressed at the *page* level, each page containing between 512 byte and 2 KB. Additionally, each access suffers from a relatively important latency ($20\mu s$), three orders of magnitude higher than SRAM or DRAM (but still three orders of magnitude better than an HDD).

3.2 The ReMIX Architecture

The ReMIX system is based on a PCI board which integrates a Xilinx Virtex-II Pro FPGA coupled to 64 GB of NAND-Flash memory. This memory is organized in multiple parallel banks as illustrated in Fig. 2. Our prototype system is fully operational, and several applications in the field of bio-informatics have already been successfully ported to it [8]. A simple file system allows the user to transparently manage the content of the 64 GB Flash memory, while guaranteeing optimal throughput when accessing data during processing stage.

Porting an application to the ReMIX machine consists in designing a hardware filter which follows a simple data-driven FIFO interface. The data throughput at the filter input (e.g. at the Flash output) is approximately 640 MBps, while the filter output (e.g the host PCI bus) throughput is restricted to 5 MBps (the target PCI board only supports slave I/O). The following Section describes our hardware filter architecture, and how we accounted for these constraints during its design.

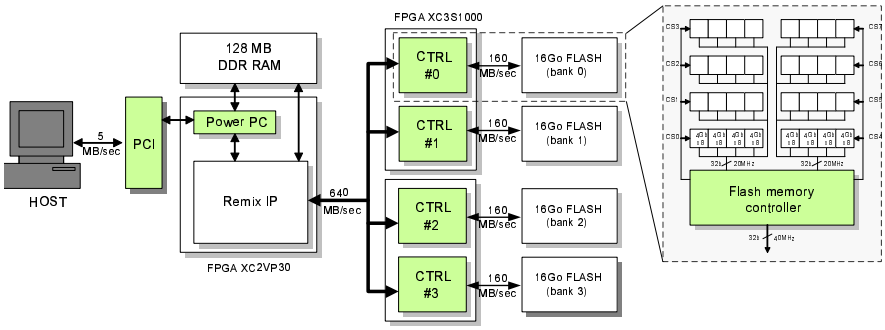

Fig. 2. RMEM card architecture

4 A Hardware Filter Architecture for CBIR

Profiling data shows that the most time-consuming step in the CBIR algorithm is the distance computation: it takes more than 98 % of the total execution time. This is not surprising: searching an image database containing B descriptors with a query image from which Q query descriptors are extracted requires $B.Q$ distance computation steps. It therefore seems natural to try to speed-up this part of the application with a dedicated hardware architecture.

4.1 Accelerating Distance Computation

The distance computation algorithm can be seen as a triple nested loop with data dependencies limited to the most inner loop (distance accumulation). This algorithm can be very easily parallelized as a 2D-systolic architecture. However, such an architecture requires accessing 24 descriptor components per cycle, while the Flash memory can only produce 8 bytes per cycle when the hardware filter is clocked at 80 MHz. Instead of this pure systolic implementation, we use a partitioned systolic linear array which is represented in Fig. 3. This architecture allows for parallel distance computation between Q fixed query descriptors q_i and every single descriptor b_j of the database, the B database descriptors being read from the Flash memory.

When a database descriptor b_j is read, distances are computed and accumulated for each descriptor component and eventually, a distance $d(b_j, q_i)$ is computed for each query descriptor q_i. In the initial software implementation of our algorithm, distances were computed using floating-point arithmetic. However, floating-point in FPGAs has major drawbacks in terms of performance and resources usage [6,4]. We have shown in [12] that using 8-bit fixed-point arithmetic for descriptor distance computation preserves the accuracy of the results. Similarly we have replaced euclidian distance by Manhattan distance since the latter allows multiplication to be replaced by a simple compare-and-add instruction. Table 2 summarizes resource usage and performance (in MHz) of a single array Processing Element for various bitwidth.

The throughput of a linear processor array of 24 PE is 74 MBps: in other words it processes a new descriptor approximately every 26 cycles, and produces 24 distance scores every 26 cycles.

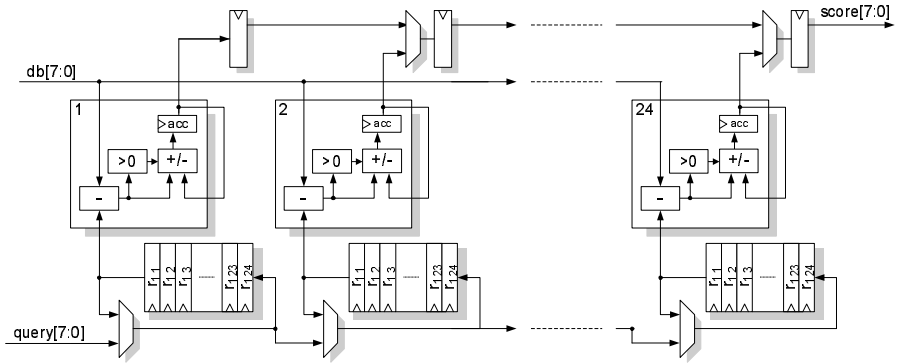

Fig. 3. Distance computation component

Bitwidth	24 bits	16 bits	12 bits	8 bits	3 bits
Resource (Slices)	62	42	32	22	11
Frequency (MHz)	147	161	161	161	168

Table 2. Resource and performance for a distance computation PE as a function of bitwidth

As we now accelerate the distance computation, it is the selection which is likely to become a performance bottleneck. Indeed, because of the limited output bandwidth available at the filter IP block output port, it is not reasonable to forward all distance scores to the embedded processor or to the host CPU, since our IP would be slowed down by I/O stalls. We thus propose to implement the selection entirely in hardware, so that it can be integrated within the filter IP. With this approach, the filter simply forwards the content of the k-NN lists to the host for the *election step*.

4.2 Accelerating Selection using Hardware

The selection consists in sorting distance scores and in retaining the k best distances. In the software implementation, selection is implemented as a periodic sorting: all distance scores that may be part of the final list, – that is to say, all the distance scores that are below at least one item of the current list – are stored in a buffer. Once full, this buffer is sorted, and is merged with the previous list to form the updated list. This approach is very efficient in practice and profiling data show that selection represents less than 1% of the total execution time.

Implementing sorting in hardware is a well-studied problem, and several highly parallel solutions have been proposed, ranging from sorting networks [2] to systolic arrays.

- Sorting networks are very efficient to sort data that enter the sorter in parallel: networks structure of spatial complexity $O(n \log n)$ can sort n tokens every cycle.
- Systolic sorting is more suited to sort data-streams that enter the processor array sequentially. Sorting is then done in $O(n)$ time on a systolic array with n processors.

As distance scores are produced by the distance computation component at a rate of one score per cycle, selection must be done on the fly. Moreover, we are only interested in the k lowest distance scores, where k is very small as compared to the total number of scores n produced by the distance computation step. These observations suggest that neither sorting networks nor systolic sorters are appropriate solutions. One could think of using a modified systolic sorting array with only k processors, however this approach still requires important hardware resources. Another important observation is that, once the steady regime is reached, and since $n \gg k$, only very few distance scores would walk past the first processor of a systolic sorter array[5]. This would result in a highly inefficient architecture in which processors would remain idle most of the time.

Instead of using systolic sorting, we therefore propose to implement selection as a simple insertion sorting. Fig. 4 represents the insertion sorting datapath, which consists of a dual-port on-chip memory associated to a simple comparator. This architecture can handle up to 32 lists, which is enough since the output flow of our distance stage consists in distance scores related to 24 distincts query descriptors.

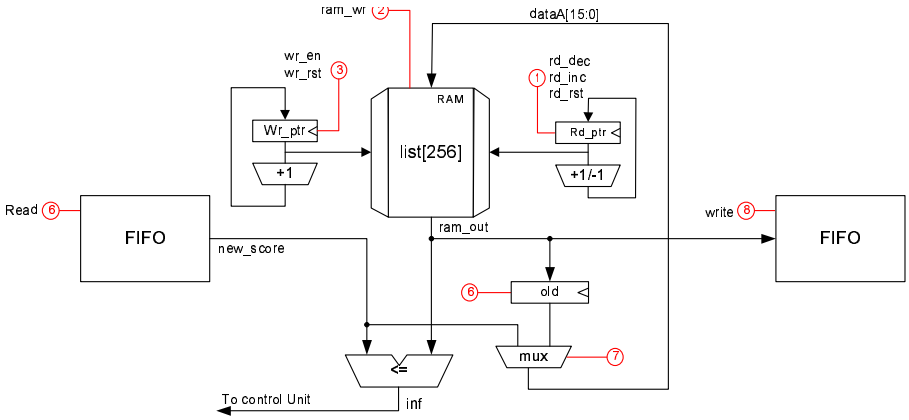

Fig. 4. Structural view of the sorting element

This solution is highly inefficient from a theoretical point of view, since its time complexity for obtaining the k-NN from a set of n distance scores is $O(n.k)$. However, its practical complexity remains very close to $O(n)$: the vast majority of scores (more than 99%) are not to be inserted into the k-NN list and would just pass through the insertion sorting step with an overhead of a few cycles. Even so, this overhead is still unacceptable, since it happens for each distance score. To remove this overhead, we perform a preliminary filtering step which buffers potential matches into a FIFO, as described in Fig. 5.

[5] Note that the software implementation takes advantage of this property to skip distance calculation whenever the current score is above its corresponding k-NN list threshold score

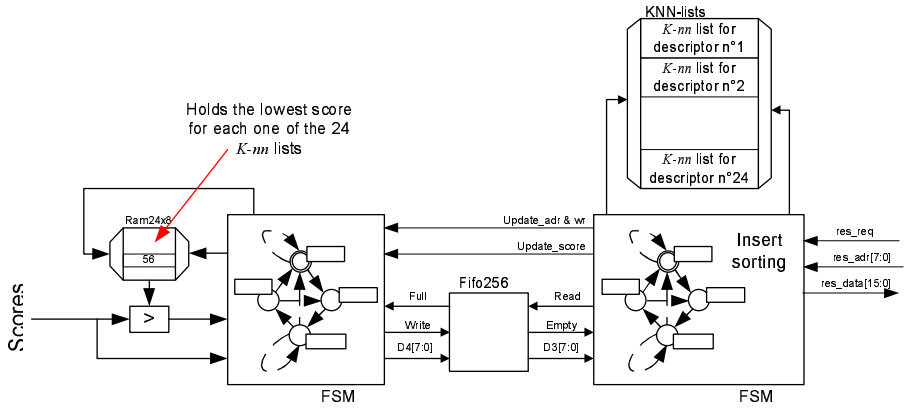

Fig. 5. Structural view of the selection component

The resulting selection component uses 135 FPGA slices and one RAM block and its maximum operating frequency is 200 MHz. It allows each new score to be inserted at the n-th position in the k-NN list in $n + 3$ cycles.

4.3 Putting it Altogether

The Xilinx Virtex-II Pro FPGA used in the ReMIX machine has 15,000 slices and approximately 50% of them are used by the ReMIX controller. It is therefore possible to take advantage of the available space by implementing several *lines* of distance computation and selection units which operate in parallel, as shown in Fig. 6. In this execution scheme, each line is given a different subset of query descriptors, and they all process the same database subset in parallel.

Each line contains 24 query image descriptors. Therefore, each time the entire database is read, the filter computes in parallel the nearest neighbors of 240 query descriptors, i.e. in average $1/3$ of an image descriptors set. When the whole database has been processed, the lists of query descriptors are flushed out separately through a simple pipeline mechanism.

Table 3 compares various bit-width implementations, in terms of maximum number of lines, with the maximum frequency and slices usage for each case. This shows that with 8 bits descriptors and a ReMIX architecture clocked at 80 MHz, 10 such lines can be instantiated.

5 Experimental results

In this section, we present the performance model that guided our design choice, and then compare the estimation obtained from this model to the actual performance results.

[6] Given 10 lines, any frequency optimization overflows the number of available slices.

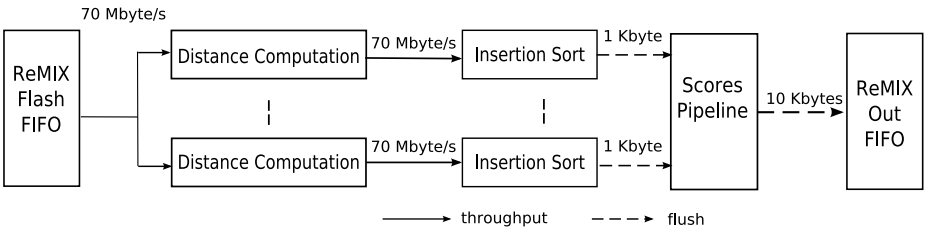

Fig. 6. Filter as integrated in ReMIX

Bitwidth	24 bits	16 bits	12 bits	8 bits	3 bits
Nb lines	4	5	7	10	16
Resource (Slices)	7037 (98%)	6206 (93%)	6865 (97%)	7180 (99%)	7078 (99%)
Frequency (MHz)	124	129	131	84[6]	125

Table 3. Resource and performance for the maximum number of processor lines as a function of bit-width, with Synplify used for synthesis and Xilinx ISE8.1 used for back-end

5.1 Theoretical Performance Model

Let N_d be the number of descriptors in our database, T_c the filter clock period (set at 12.5 ns in our tests), N_{PE} the total number of distance processor in the design (here $N_{PE} = 240$) and k the number of items in each k-NN list (we have $k = 32$). Knowing that our implementation of distance computation proceeds at the rate of 1 descriptor every 26 cycles, the time T_{calc} required to perform the distance computation step is given by $T_{calc} = 26 N_d T_c$. However, our performance model must also account for several performance overheads.

For example, the whole descriptor database cannot be read from the Flash memory in a single pass, it is therefore necessary to split the database into N_c chunks. This induces a chunk access overhead T_{chunk} which value can only be determined experimentally (our performance model will therefore ignore this overhead).

On the other hand, whenever a FIFO reaches its maximum capacity, the distance computation component must be stalled to avoid data-losses. Quantifying the overhead due to these stalls (we write T_{stalls} this overhead) would require to have a precise model for the FIFO usage over execution time. However, a simple probabilistic reasoning can help us to obtain a higher bound for T_{stalls}.

Let us consider a scenario in which there is no FIFO between the pre-filtering step and the insertion sort. In such a situation, each *match* causes the distance computation component to stall until the *match* is correctly inserted in its corresponding k-NN list.

We write p_{match} the probability for a distance score to not be pre-filtered before insertion. We know from software profiling that this value is close to $p_{match} = 2.10^{-5}$. We also know that our insertion sort component has a worst-case execution time of $k+3$ cycles. The percentage of stall cycles in this scenario can therefore be bounded by:

$$p_{stall} = \frac{p_{match}}{p_{match}(k+3)+1}$$

The overhead (that we can write as $T_{stall} = p_{stall} T_{calc}$) in this scenario can be considered as a higher bound of what we would observe in practice, since we use a 256-slot FIFO buffer between pre-filtering and sorting steps.

Finally, we must also account for the time spent flushing out the k-NN lists (we write T_{flush} this overhead). This step has a fixed impact on the global execution and can be written as :

$$T_{flush} = k.\frac{N_{PE}}{B_{PCI}}$$

where B_{PCI} stands for the actual output PCI bandwidth (here we have $B_{PCI} = 5\,\text{MBps}$). The total search time T_{search} for a query image containing N_q descriptors can then be written as :

$$T_{search} = \left\lceil \frac{N_q}{240} \right\rceil (T_{calc} + T_{stalls} + T_{flush})$$

5.2 Measured and projected performance

We have benchmarked our design over a real life 30,000 images database consisting in 20,868,278 descriptors. This database normally requires 2 GB of storage, however, thanks to the use of 8 bit fixed point arithmetic instead of single precision floating-point this size was reduced to 650 MB.

Using the performance model obtained, we estimate that searching a 30,000 image database with a 720 descriptors query leads to a search time of $T_{search} = 19,68\,s$. Running the same search on the actual ReMIX system lead us to an observed search time of 20.43 seconds, that is within a 4 % error margin of the predicted performance.

The actual speed-up factor over the original software implementation is 45. In other words a single ReMIX system is as efficient as a 45 PCs cluster. While this acceleration factor only holds for descriptors encoded as 8 bits integers, we estimated the corresponding results for different bitwidth by implementing as many processor lines as possible on the FPGA and use this result to estimate the corresponding speed-up. These performance projections are summarized in Table 4.

Bitwidth	24 bits	16 bits	12 bits	8 bits	3 bits
Query Time (sec)	50	40	28.5	20	12.5
Speed-up factor	18	22	31	45	72
Number of lines	4	5	7	10	16

Table 4. Estimated search time and corresponding speed-up for varying bitwidth for a 30.000 images database

6 Conclusion

In this paper we have proposed a hardware accelerator for Content Based Image Retrieval based on local descriptors. Our architecture performs both the distance calculation and selection in hardware, and was experimentally validated on the ReMIX machine using a real-life image database.

A single PCI board associated provides similar performance to a 45 node PC cluster, at $1/20$ the price (without accounting for the host PC). An interesting point is that our hardware architecture can easily be targeted at different descriptors (e.g. for different precision, or higher number of dimension), with moderate design efforts.

Directions for future work include studying the implementation of CBIR on GPUs which appear to be an interesting platform to speed-up this type of algorithms.

References

1. Laurent Amsaleg and Patrick Gros. Content-Based Retrieval using Local Descriptors: Problems and Issues from a Database Perspective. *Pattern Analysis and Applications*, 2001.
2. K.E. Batcher. Sorting networks and their applications. In *Proceedings of the AFIPS Spring Joint Computer Conference 32*, 1968.
3. Christos Faloutsos. *Searching Multimedia Databases by Content*. Kluwer Academic Publishers, 1996.
4. Gokul Govindu, Ling Zhuo, Seonil Choi, and Viktor Prasanna. Analysis of High-performance Floating-point Arithmetic on FPGAs. In *Reconfigurable Architecture Workshop*, 2004.
5. Stéphane Guyetant, Mathieu Giraud, Ludovic L'Hours, Steven Derrien, Stephane Rubini, Dominique Lavenier, and Frédéric Raimbault. Cluster of Reconfigurable Nodes for Scanning Large Genomic Banks. *Parallel Computing*, 2005.
6. Walter B. Ligon III, Scott McMillan, Greg Monn, Kevin Schoonover, Fred Stivers, and Keith D. Underwood. A Re-evaluation of the Practicality of Floating-Point Operations on FPGAs. In *FCCM '98: Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, 1998.
7. L. Kostoulas and I. Andreadis. Parallel Local Histogram Comparison Hardware Architecture for Content Based Image Retrieval. *Journal of Intelligent and Robotic Systems*, 2004.
8. Dominique Lavenier, Xinchun Liu, and Gilles Georges. Seed-based genomic sequence comparison using a fpga/flash accelerator. In *To appear in Proceedings of EEE International Conference on Field Programmable Technology*, 2006.
9. Herwig Lejsek. A case-study of scoring schemes for the pvs-index. In *CVDB '05: Proceedings of the 2nd international workshop on Computer vision meets databases*, pages 51–58, New York, NY, USA, 2005. ACM Press.
10. Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(10):1615–1630, 2005.
11. Koji Nakano and Etsuko Takamichi. An Image Retrieval System Using FPGAs. In *Proceedings of ASPDAC*, 2003.
12. Auguste Noumsi, Steven Derrien, and Patrice Quinton. Acceleration of a content-based image-retrieval application on the RDISK cluster. In *International Parallel and Distributed Processing Symposium (IPDPS 2006)*, 2006.
13. David Patterson, Thomas Anderson, Neal Cardwell, Richard Fromm, Kimberly Keeton, Christoforos Kozyrakis, Randi Thomas, and Katherine Yelick. A case for intelligent ram. *IEEE Micro*, 17(2):34–44, 1997.
14. Oscar D. Robles, José L. Bosque, Luis Pastor, and Angel Rodríguez. Performance Analysis of a CBIR System on Shared-Memory Systems and Heterogeneous Clusters. In *IEEE International Workshop on Computer Architectures for Machine Perception (CAMP'05)*, 2005.
15. Constantinos Skarpathiotis and K.R. Dimond. A Hardware Implementation of a Content Based Image Retrieval Algorithm. In *International Conference on Field Programmable Logic and Application (FPL)*, January 2004.