Acceleration of a Content-Based Image-Retrieval Application on the RDISK Cluster *

Auguste Noumsi[1], Steven Derrien[2], Patrice Quinton[3]

[1]IRISA/Université de Douala
35042 Rennes, France
noumsi@irisa.fr

[2]IRISA/Université de Rennes 1
35042 Rennes, France
sderrien@irisa.fr

[3]IRISA/ENS Cachan
35042 Rennes, France
quinton@irisa.fr

Abstract

Because of the growing use of multimedia content over Internet, Content-Based Image Retrieval (CBIR) has recently received a lot of interest. While accurate search techniques based on local image descriptors exist, they suffer from very long execution time. We propose to accelerate CBIR on the RDISK machine, a cluster of FPGA-enhanced hard-drives, that follows the philosophy of smart-disks. Our platform combines coarse and fine grain parallelism thanks to the concurrent use of the cluster nodes and of a programmable logic device. The implementation of the CBIR application on this mixed hardware/software platform follows a strict methodology, that was validated on realistic data-set (image database of more than 30,000 images). This methodology allows us to adapt the original algorithm to suit a hardware implementation, and to select the values of some key design parameters to maximize global performance. Our preliminary results indicate that speed-ups between 120 and 200 could be obtained for a cluster of 32 nodes compared with a software implementation running on a standard desktop PC.

1. Introduction

Content Based Image Retrieval (CBIR) is a technique that allows one to find out images of a database that are (at least) partly similar to a given reference image. CBIR is drawing increasing interest due to its potential application to problems such as image copyright enforcement. Indeed, the large use of Internet resulted in a huge increase of Web-available multimedia content, especially images. Checking copyright is therefore a concern for image owners, who must be able to identify undue use of images. This identification process relies upon precise and fast image-comparison algorithms, as Internet is a rapidly changing medium, and such algorithms need to be run on a daily basis.

1.1. CBIR Systems

CBIR is mainly based on the comparison of *image descriptors* of a *reference image* with those of a *descriptor database*. Descriptors may be either *global*, i.e. they represent some global feature of an image (e.g. a grey-level histogram) or *local*, in which case they describe special points of interest in the image (e.g. corners, color changes, etc.). In this paper, we are concerned with local descriptors only: recent research has shown that they provide a more robust approach since they are less dependent to image variations than global descriptors.

Retrieving an image consists first in associating a set of descriptors with the reference image – typically, a few hundred vectors of 24 real components, – then in computing the distance between each one of these descriptors and those of the database images (*distance calculation stage*). For each reference descriptor, a k-nearest neighbor sorting selects the k database descriptors whose distances are the smallest (*selection stage*). Finally, votes are assigned to the images depending on their occurrences in the k-nearest neighbor lists (*election stage*): the image that has the largest number of votes is considered to be the best match. The whole process is extremely time consuming: retrieving an image in a 30,000 image database requires about 1,500 seconds 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

*This research was partly funded the French Ministry of Foreign Affairs as part of Sarima project no. 2002-84.

1-4244-0054-6/06/$20.00 ©2006 IEEE

to definitive results because of a phenomenon called *dimension curse* that affects large databases operating on higher-dimensional data sets [2, 7].

Therefore, the only solution to improve the performance of current CBIR systems is to accelerate the algorithm using special-purpose implementations. Accelerating the application on a parallel machine is the most natural choice, and has already been studied by few authors, among whom Robles et al [18].

In this paper, we approach this problem by combining both parallelism and special-purpose hardware. Our model architecture is that of a so-called *smart disk*, whose prototype is RDISK [9], implemented and available at IRISA. In short, a *smart disk* is a collection of disk drives, each one controlled by a reconfigurable, FPGA-based processor, called a *filter*. Filter processors handle on the fly data coming out from the disk that they control and transmit those data that are relevant to a given request to a common host processor through a low-bandwidth network. The host processor then performs a final combination algorithm in order to get the results.

1.2. Related Work

The poor performance of current CBIR systems is recognized by the CBIR community as an important problem that limits the spread of these techniques. However, as acknowledged by Datta et al [5], only limited efforts were put in that direction. While a few papers report parallel implementation of CBIR on shared memory machines and PC clusters [18], there has been almost no work on special purpose hardware for this application domain. Moreover, none of the few noticeable exceptions address the problem in the context of a *real-life* hardware system (e.g with its communication interface, I/O bandwidth constraints, etc.) [11, 19, 14].

The remaining of this paper is organized as follows. Section 2 presents the RDISK platform, from both the system level and architectural point of view. Section 3 briefly describes the problems that needed to be addressed to obtain an efficient implementation, and the methodology that we used to solve them. Section 4 presents some of the algorithmic transformations we used (floating-point to fixed-point format conversion, and distance metric change), and Section 5 focuses on the derivation of the hardware filter architecture. Section 6 describes the performance model that served as a baseline for all our design choices, and the optimization problems resulting from this modeling. Preliminary results are given and discussed in Section 7, and conclusion and future work directions are then sketched.

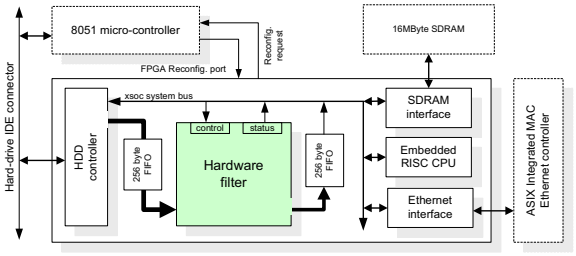


Figure 2. The RDISK System on Chip Internal Organization.

2. The RDISK Cluster Architecture

RDISK is a reconfigurable *smart disk* cluster research prototype that has been developed at IRISA since 2001 and that borrows from the *smart-disk* concept. Although the concept of such processing capable storage devices dates back to the 70s [3], it has drawn an increasing attention at the end of the 90s [17, 10, 12, 1] thanks to the emergence of Storage Area Network (SAN) and Network Attached Storage (NAS). The basic idea of a *smart disk* is to take advantage of the often underused computing power that is available on disk controllers, in order to implement some of the search operations directly on data as they flow out of the disk. RDISK elaborates on this concept by building an architecture where each disk controller is a reconfigurable FPGA-based processor.

The RDISK system [9] was designed to provide high performance with low-cost hardware components. Our initial goal was to build a system whose nodes would cost no more than one tenth of a PC cluster (i.e. approximately $200). RDISK consists (see Fig. 1) in a number of *nodes* – typically, a few tens; a node contains a 40 GB IDE disk, a 100 Mbps Ethernet controller, 16 MB of SDRAM, a 8051 micro-controller, and an FPGA chip that serves as the *filter processor*.

The filter processor allows data coming from the disk to be processed *on the fly* in order to select those data that are relevant for some application algorithm, e.g. a search query. Filtered data are then sent to a *post-processing host* through a 100 Mbps Ethernet switch.

Fig. 2 depicts the internal organization of the RDISK FPGA. The chip is a Spartan-II FPGA from Xilinx, and it offers an equivalent density of 200,000 logic gates. It is used to implement a generic System on a Chip design, the fixed part of which includes a hard disk-drive controller, a 16-bit RISC embedded CPU, and an Ethernet chip-set interface (for more details, see Guyetant et al. [9]). The reconfigurable (i.e. the application specific) element constitutes

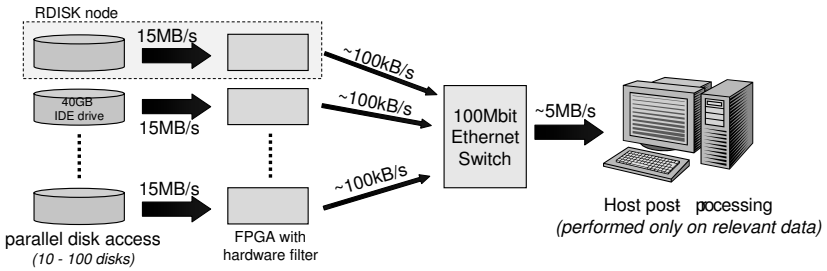


Figure 1. The system level view of an RDISK cluster

the hardware filter component. This filter is designed according to the template interface described in Fig. 2. Its rôle is to process in real-time the data stream coming from the disk drive and to send its output to the embedded CPU. This embedded CPU then performs a post-filtering process and sends the final results to the host through the Ethernet network. Programming an RDISK node thus consists in designing an *application dependent hardware filter* (in VHDL, Verilog, or higher level languages such as Handel-C), in writing a *post-filtering C program* to be run on the embedded CPU, and in writing a *post-processing program* for the host processor.

An important point of RDISK is its reconfigurability: at any time, upon receipt of a specific command from the host, an RDISK node can switch to another hardware configuration and its associated hardware filter. This reconfiguration is handled by the 8051 micro-controller which reconfigures the FPGA from a bit-stream file stored on the hard-disk. Each node can store up to 256 hardware configurations, and a custom file system allows configurations to be added or removed from the drive by the host.

The RDISK cluster thus takes advantage of two levels of parallelism: coarse grain parallelism through the concurrent use of the nodes, and fine grain parallelism within each hardware filter; we shall indeed see that a large amount of fine grain parallelism can be used when designing a hardware filter for the FPGA processor.

Our RDISK cluster prototype is fully functional and has already proved to be a very promising platform: two applications of computing biology have been successfully implemented and speed-ups of two orders of magnitude have been reported [9].

3. CBIR Implementation on RDISK: Problems and Methodology

The basic mapping scheme of CBIR on the RDISK n-node cluster is quite straightforward. The database descriptors are equally distributed among the n nodes of the cluster. Each filter gets a fraction q_0 of the q descriptors associated to the query image and computes the distance between each one of the q_0 descriptors and a subset b_0 of the database descriptors.

As the distances are computed, they are also sorted and only the k-smallest distances are kept. These distances are then sent to the host processor where the final election stage is performed. This simple scheme raises however several questions, that we list now.

1. As the filter processor is based on the FPGA technology, efficient implementations cannot be obtained using floating-point calculations, a mere translation of the initial software description. Therefore, an analysis of the calculation precision requirements is mandatory. We do it in Section 4.

2. The second question is how to efficiently implement the distance computation step on the FPGA. Given that most FPGA design operate at frequencies below 100 MHz, the only way to reach good performance is to take advantage of fine grain parallelism within the FPGA. Section 5 will show how this can be done efficiently.

3. Finally, we have to solve the problem of performance at the system level. We do so by providing a detailed performance model in Section 6. This allows us to estimate global performance, and to optimize the hardware filter design.

4. Optimizations for Hardware Implementation

The software implementation of an algorithm often requires important modifications when it is to be implemented as application specific hardware. In this section we present the two main transformations that we applied on the initial CBIR specification: the conversion from floating-point to fixed-point arithmetics, and the use of the L_1 distance as an alternative to the Euclidian (L_2) distance.

4.1 Using Fixed-Point Arithmetics

While floating-point has very good hardware support on modern CPUs, it is very poorly suited to an FPGA implementation. Although floating-point operators can be implemented in programmable logic [8], their realization is very costly in terms of resource usage, and they provide only limited performance compared to most fixed-point arithmetics implementations. For instance, a single precision floating-point adder requires 80 times the area of a 8-bit fixed-point adder. As far as CBIR is concerned, encoding the descriptors using an 8-bit fixed-point format also reduces the database size by a factor of almost 4. This in turn speeds up the database scanning time. The ability to use fixed-point arithmetics should hence help us to increase, thanks to higher parallelism and I/O bandwidth, the overall level of performance of our hardware implementation.

4.2 Conversion Methodology

Moving from floating-point to fixed-point is not straightforward. Such a conversion generally induces a loss of precision in the computations (due to quantization and rounding errors) which may in turn impact the Quality of Results (QoR) of the algorithm. Using analytical models, it is possible to quantify, in terms of Signal to Quantization Noise Ratio (SQNR), the impact of a conversion to fixed-point format [13]. However this modeling is useful only when it is possible to determine an upper bound of the acceptable SQNR for the application at hand (this is very often the case in signal processing applications).

Unfortunately, in the context of CBIR, there is no way to directly relate the SQNR to the quality of the search results. The only solution is hence to use extensive simulation and observe experimentally the impact of a conversion scheme on the search results. Ultimately, we expect to determine the narrowest fixed-point encoding that will preserve the CBIR accuracy. To do so, we first defined a metric for quantifying the CBIR answer accuracy, and then we explored various possibilities of fixed-point encoding to efficiently combine *scaling* and *saturation*.

Fixed-point encoding consists in mapping any value r belonging to $\mathbb{R}$ to an signed integer domain $[-B, B-1]$ where $B = 2^{w-1}$ and w is the targeted fixed-point format bitwidth. This conversion uses two parameters: the domain bound B and the target bit width w. Let $Q(r)$ denote the converted value. This conversion is done according to the expression:

$$Q(r) = \begin{cases} \lfloor 2^{w-1} r \rfloor & \text{if} \quad r \in [-B,\, B-1] \\ B-1 & \text{if} \quad r \geq B \\ -B & \text{if} \quad r < -B \end{cases} \tag{1}$$

From the initial data distribution histogram, represented in figure 3, one can observe that this distribution is concentrated within a very narrow interval; this suggests that mapping the initial dataset to a short interval will have limited impact on accuracy. In the following, we choosed the mapping interval such that 97% of the initial descriptor components would belong to it.

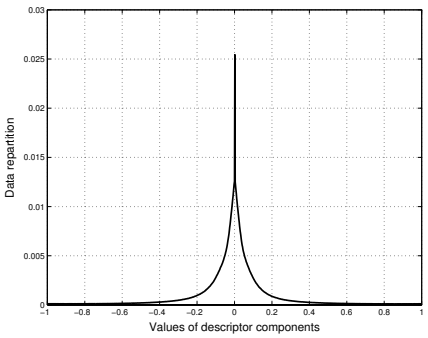


Figure 3. Descriptor component distribution histogram

4.3 Changing the Distance Metric

Another typical transformation when dealing with a hardware implementation is *strengh reduction*. It consists in replacing a costly operation (say multiplication, or division) by a simpler one (usually addition or shift) that is functionally equivalent to the initial operation. In this work, we performed a somewhat similar transformation: we proposed to substitute the standard Euclidian (L_2) distance by the Sum of Absolute Difference (SAD, L_1) distance. This allows the square-accumulate operation to be replaced by a simple substract-accumulate with much lower resource usage. Note that this transformation does not lead to a functionally equivalent implementation. Therefore, its effect had to be checked by simulation.

4.4 Validation Methodology

To perform this validation, we used an accuracy test that is based on the work of Amsaleg et al [2]. We consider a random image I_{ref} taken from the image database. From this image, we derive a set of image variations (including I_{ref}) using a set of transformations (cropping, rotation, JPEG encoding, etc.) taken from the Stirmark benchmark [15]. Each of these images is then used as a query for the database.

According to the CBIR algorithm, the *election* step results in a list of pairs (I_i, s_i) in which I_i corresponds to an image of the database and s_i to its score (i.e. the number of

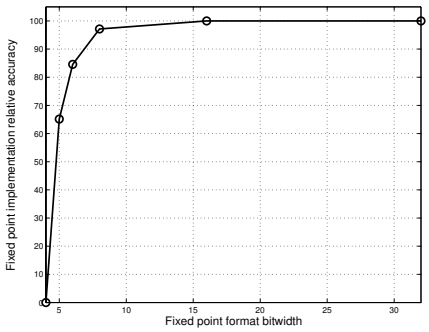


Figure 4. Relative accuracy for various fixed point encoding format

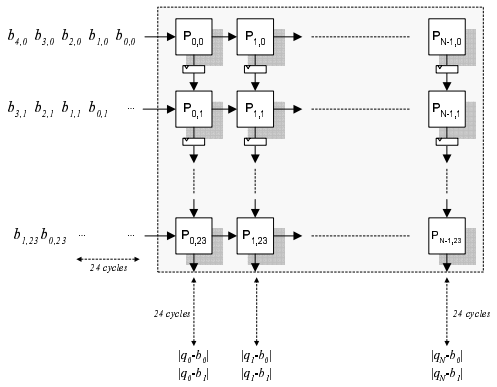


Figure 5. A 2D systolic array for distance computations

votes received by the image). This list is sorted according to the number of votes, I_1 beeing the image with the higher score. The *accuracy* of the search is then defined by:

$$\mathrm{Acc}(I_r) = \begin{cases} 1 & \text{if } I_1 = I_{ref} \text{ and } s_1 > 2s_2 \\ 0 & \text{otherwise} \end{cases} . \quad (2)$$

To determine the global accuracy, we computed the average accuracy score for a set of query images containing several hundreds of images. This test corresponded to several weeks of computation on a standard workstation.

4.5 Results

Figure 4 shows the relative accuracy obtained for various fixed-point bit width format using the L_1 distance compared to the original floating-point implementation using the L_2 distance. For bitwidths of 8 and above we obtain results which are almost identical to those of the original software implementation (whose global accuracy is 85%). We can hence take advantage of this information during the hardware filter design stage to reduce the resource usage and increase the implementation performance.

5. Parallel Implementation

In this Section, we turn to the problem of implementing efficiently distance calculations using the fine-grain parallelism that is available on the FPGA.

5.1. Deriving a Parallel Architecture

Our approach is inspired from systolic-array design methodologies [16] and especially partitioning techniques. The parallelization methodology in itself is out of the scope of this paper: we will therefore limit ourselves to show its results (e.g. the parallel processor arrays on which the distance are computed) and provide an intuitive explanation.

Fig. 5 depicts a straightforward 2D systolic architecture for the distance computation. A network of $24 \times N$ elementary processors, where N is the number of reference descriptors, allows each distance to be computed in a pipeline fashion, from the top to the bottom: each column of this array computes the distance between a database descriptor $\{b_{k,m}\}_{0 \leq k \leq 23}$ that is input to the left of the array, and one reference descriptor. The resulting distances appear at the bottom of the array.

A quick evaluation showed that this architecture is not suited to RDISK. First, it uses too much logic resource to be considered for the actual Spartan II FPGA chip. Second, the bandwidth necessary to feed this processor array is about 600 M descriptors per second for a hardware filter running at 25 MHz, which is far beyond the 15 MBps available from the disk drive.

5.2. Partitioning the Architecture

A *partitioning* transformation of this architecture allows one to adjust the resources and the bandwidth of the hardware filter to the RDISK architecture. The Locally Sequential Globally Parallel (LSGP) partitioning scheme (see [6]) consists in grouping together processors into so-called *tiles*, and to merge these tiles. Computations are then executed sequentially for each tile by a unique processor. For example, Fig. 6 shows the architecture of Fig. 5 after LSGP partitioning using a 24×3 tile. This new architecture has therefore $\frac{N}{3}$ physical processors: one processor takes care of 3 successive columns of the initial network. Thus, it reads an average of one word of a reference descriptor every 3

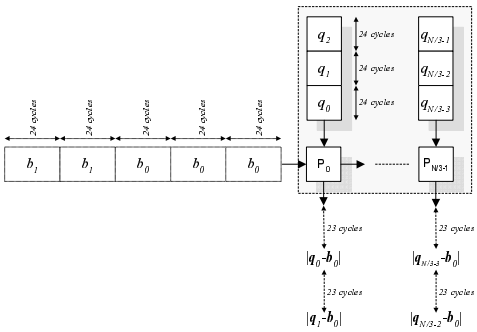


Figure 6. Equivalent 24×3 partitioned processor array

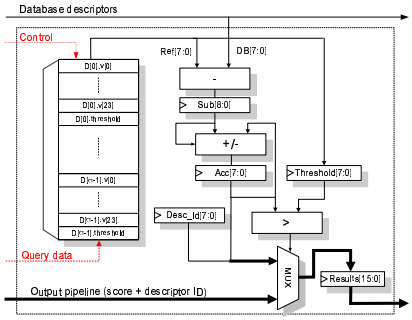


Figure 7. The filter processing element architecture and its 3 stage pipelined datapath (control signals are not represented)

cycles. As a counterpart, it needs to store three complete descriptors, which increases its memory resource cost.

Note that this partitioning results in a linear array, since the first dimension is equal to the height (24) of the array. Doing so has several advantages: it simplifies the hardware interface with the hard-disk FIFO, and it allows further optimizations that will be described in Subsection 5.3. From now on we will hence only consider tiles of the form $24 \times \sigma$.

In addition to this transformation, we also pipelined the internal structure of each processor using *cut-set retiming*, as shown in Fig. 7: combining this with partitioning provides very efficient implementations [6].

5.3. Sorting and Thresholding

As seen in Section 5, the goal of the hardware filter is to find out, for each reference descriptor, a list of k database descriptors that provide the best k distances. This requires distance calculation to be followed by a sorting. Implementing sorting as a fine grain parallel architecture would be possible (for example, using a sorting systolic array), but it would be resource consuming and inefficient.

We therefore made the choice to implement this sorting stage as part of the *post-filtering* step on the embedded CPU. In this procedure, the data for post-filtering are transmitted to the post-processing host only once the node has finished scanning its local database.

One important assumption of this design choice is that we expect the *post-filtering* not to be a performance bottleneck. In our case, since no communication occur between the host and the node during the scan, the network processing workload on the embedded CPU is very limited. This leaves almost all its processing power for performing this *post-filtering* step. We also know (from benchmarking) that our embedded CPU is able to sustain the sorting and insertion of at least $65\,10^3$ distance scores per second (with worst case insertion complexity).

In the original distance computation processor array, each database descriptor produces as many distance scores as there are query descriptors stored in the array. This leads to a throughput far above the CPU processing capabilities. To reduce this throughput, we use the fact that the distance of the k-th element of the distance list constitutes a natural threshold for the distance calculation processors: any partial accumulated distance that exceeds this threshold will not appear on the final distance list, and can therefore be discarded. As a consequence, the filter only outputs those database descriptors, whose scores are below at least one of the query descriptors (we call such a descriptor a *match*).

Using execution traces, we observed that the average probability for a database descriptor to be a match for a given query descriptor is $p = 1.89\,10^{-5}$. From there, we derived an estimation of the actual filter selectivity which depends on the number of query descriptors handled by the filter (the more query descriptors the higher the chance to have a match). This lead to an average of $10.67 n_d$ matches per second (with n_d beeing the number of query descriptors handled by the array). In other words, the CPU is able to sustain post-processing for a hardware filter handling up to several thousands query descriptors.

Additionally, if processor say P_n, while computing the distance between reference descriptor number n and some database descriptor, detects that the accumulated distance exceeds this threshold, it can ignore the remaining of the database descriptor and immediately jump to the next one, hence saving computation time. In the rest of this paper we call *threshold overflow* such an event.

The threshold overflow optimization is currently used in the software implementation of the CBIR application, and it has proved to be very efficient, since it reduces the computation volume by 8 and the execution time by a factor

of 4. Unfortunately this optimization is not that efficient when used in our parallel architecture. Because it behaves as an SIMD architecture, the array cannot proceed to the next descriptor unless all processors have overflowed their corresponding threshold. This observation has a severe impact on the actual efficiency of our architecture: the more we add processors to the array, the longer we have to wait, as synchronization is done on the worst case.

A quantitative analysis of this phenomenon can be derived using a simple probabilistic model. Fig. 8.(a) shows the probability $P_{thr}(i)$, that during a distance computation, a threshold overflow occurs at iteration i, that is to say, after reading the i^{th} component of the descriptor. One can observe that this probability is concentrated in the very small values of i. Let now $P_{thr}(i,p)$ be the probability, for an array of p processors, that a threshold overflow occurs at iteration i, then:

$$P_{thr}(i,p) = \left(\sum_{k=1}^{i} P_{thr}(i)\right)^{p} - \left(\sum_{k=1}^{i-1} P_{thr}(i)\right)^{n} \quad . \quad (3)$$

From (3), we can easily derive the average number of iterations $\phi(p)$ that are performed by an array of p processors:

$$\phi(p) = E[P(p,i)] = \sum_{k=1}^{24} P_{thr}(p,i) \times i \quad (4)$$

Figure 8.(b) shows how $\phi(p)$ evolves when p grows. It can be observed that $\phi(p)$ grows very fast for small values of p.

This suggests that for very small values of p, adding a processor to the array brings almost no performance benefit, since the positive effect of this additionnal processing power on the execution time is annihilated by the synchronization overhead it induces.

6. System Level Optimization

Although we now have a relatively accurate architectural model, we still need to determine the set of design parameters that will allow optimal performance. Such a model is to be established at the system level, and must integrate several aspects: available hardware resource, algorithm behavior, I/O requirements, etc.

In the following, we propose such a performance model, and solve its associated optimization problem in the context of the RDISK cluster with an arbitrary number of nodes.

6.1. Modeling RDISK Node Execution Time

In this Subsection, we model the execution time by taking into account both computations and I/O timing informations.

Let us call T_{io} the time required to read a descriptor of size S_{desc} from the hard-drive, and T_{byte} the average access time for a byte on the disk, we can write $T_{io} = S_{desc}T_{byte}$. On the RDISK prototype, we have a sustained hard-drive I/O bandwidth of 15 MBps that leads to $T_{byte} = 66$ ns, and a descriptor size of $S_{desc} = 26$ bytes (24 bytes for its components plus one 16-bit word for its associated image index). We thus have $T_{io} = 1716$ ns.

Let now T_{prc} be the time required by the hardware filter to process a single database descriptor. This value depends on several parameters: the filter clock speed T_{clk}, the partitioning parameter σ, the processor pipeline depth L and $\phi(p)$ the average number of useful iterations in the distance computation loop, that itself depends on the number of physical processors p in the filter. So far p, σ, and T_{clk} have no predefined values (they are part of the design parameters). We have

$$T_{prc} = T_{clk}\left[\phi(p) + L\right]\sigma \quad . \quad (5)$$

Since I/O and computation are completely overlapped (thanks to the use of internal FIFO buffers), the actual descriptor processing time T_{calc} is defined by $T_{calc} = max(T_{io}, T_{prc})$. Since we have p physical processors working in parallel in the array, each one computing σ distance scores, the average time for a distance computation is then given by:

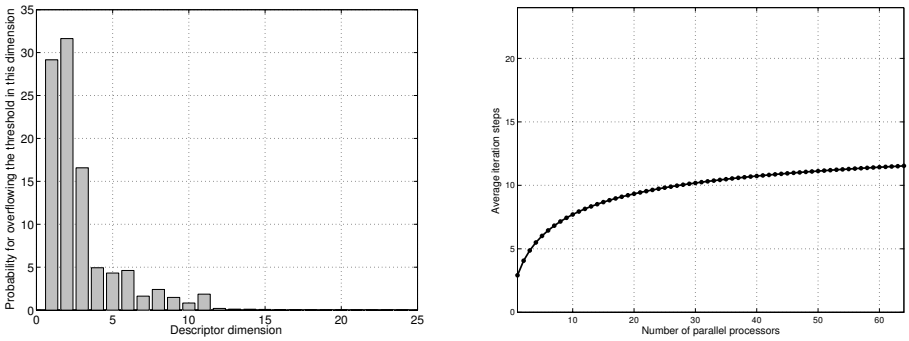
$$T_{avg} = \frac{max(T_{io}, T_{prc})}{p\sigma} \quad . \quad (6)$$

6.2. Hardware Implementation Optimization

We have to take into consideration the resource constraints of our target FPGA: number of logic cells available for implementing the elementary processor datapath, and number of memory blocks (BlockRam) available to store the query descriptors components along with their corresponding list threshold.

Our System on Chip implementation leaves 9 BlockRam, and roughly 3,300 logic cells available for the hardware filter implementation. Each BlockRam can store up to 4 kbit and can be used to implement either four 128×8-bit, two 256×8-bit, a single 512×8-bit or half a 1024×8-bit memories.

As mentioned in 5.2, the elementary processor internal memory size grows linearly with partitioning parameter σ.



(a) (b)

Figure 8. (a) probability of threshold overflow as a function of the dimension index, (b) average number of iterations as a function of the number of processors.

The memory depth of a elementary partitioned processor can therefore be expressed as $S_{mem} = 26\sigma \times 8$. Because of the rather large size of this local memory, it is likely that the most stringent resource constraint is the limited number of BlockRam. Ignoring the logic cell resource constraints, we can therefore write the maximum number of elementary processors N_{pe} that can be implemented given σ , memory block depth D_{mem}, and a higher bound on memory block resource N_{mem} as

$$N_{pe} = \left\lfloor \frac{\left\lceil \frac{26\sigma}{D_{mem}} \right\rceil}{N_{mem}} \right\rfloor \quad . \tag{7}$$

Using the model given in (6), we can now derive an estimate of the global performance as a function of the partitioning parameter σ and of the filter clock speed T_{clk}. This estimate is shown in Fig. 9. We can observe that the filter clock speed has little impact on the overall performance. This suggests that our implementation performance is rather limited by other factors: (i) the hard-drive I/O bandwidth, and (ii) the FPGA memory resources.

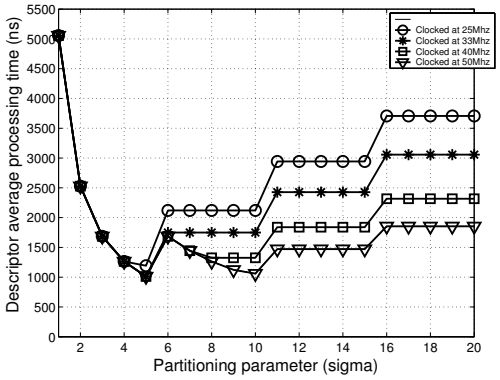


Figure 9. Performance model for various parameters performances

6.3. Modeling the Cluster Level Performance

According to the model presented previously, we can give an estimate of the average query processing time (T_{query}) as a function of N_{node}, the number of nodes in the RDISK cluster, and q the number of descriptors in the query. Assuming that each node is able to handle q_0 query descriptors, the database can then be distributed among N_g groups of nodes, where N_g is given by:

$$N_g = \left\lfloor \frac{N_{node}}{\left\lceil \frac{q}{q_0} \right\rceil} \right\rfloor \tag{8}$$

From there, knowing each RDISK node start-up time (approximately 2.5 s, caused by the FPGA hardware reconfiguration time), we can derive the average query execution time T_{query}, given a query size q and the database size B:

$$T_{query} = T_{start\,up} + \frac{B}{N_g} T_{avg} \tag{9}$$

7. Results

In this section we present some experimental results and discuss the actual benefits of our implementation.

7.1. RDISK FPGA Implementation

We have successfully implemented an RDISK SoC system that includes a hardware filter consisting of 36 elementary processors. We used SynplifyPro 7.3 as synthesis tool and Xilinx ISE 7.1 for placing and routing. This design was specified in VHDL and uses 2350 out of 2352 available slices, and supports operating frequency up to 25 MHz. Any attempt to fit a higher number of PE failed due to FPGA resource overuse (in terms memory blocks).

The hardware filter in itself occupies approximately 65% of the chip area, and the place and route software reports indicate that the filter alone can be clocked above 50 MHz. Thanks to the results summarized in Fig.8 we realized that it was not worth decoupling the hardware filter clock from the rest of the system (that operates at 25 MHz): while requiring an important design effort, such a modification would only have a very limited payoff (approximately 20% performance improvement).

So far the design has not been completely tested in a real-life situation (these tests are on the run), however the hardware filter was functionally validated at the register transfer level (using a VHDL simulator), and we expect to provide detailed performance results in the forthcoming months.

7.2. Comparing with PC Implementation

As mentioned in the introduction, a direct software implementation of the CBIR application on a 2.4 GHz Pentium 4 processor requires an average processing time of 1500 seconds for each query (with an average of 693 descriptors per query). We considered as a comparison a cluster of 32 nodes, processing queries ranging from 250 to 1500 descriptors (these are typical bounds for images). Using our performance model, we computed some estimates of the expected speed-up that are represented in Fig. 10. To summarize our results, we obtain speed-ups varying between 150 and 200 depending on the query size for the 32 node cluster. For a single RDISK node, the speed-ups vary between 4 and 6. Given that the cost of a 32-RDISK cluster can be estimated to $12,000 (including Ethernet switches, power supplies, and Rack Cabinet)[1], and assuming a cost of $400,000 for a 200 PC cluster, we can roughly estimate the price/performance ratio to 40 in the favor of the RDISK cluster.

[1]Our actual RDISK cluster has 48 nodes, its costs is approximately $15000

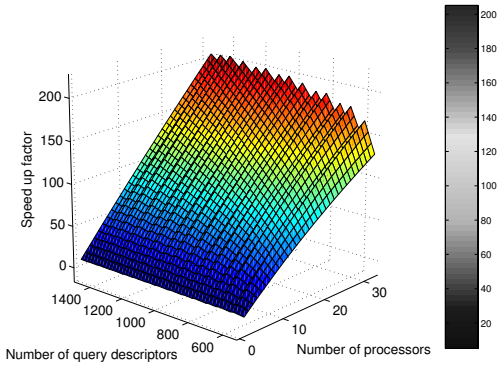


Figure 10. Speed-up as a function of the number of nodes and the number of query descriptors

On the other hand, most recent processors now integrate SIMD instruction sets targeted to multimedia applications. These instruction sets operate on short fixed-point data types (8, 16 or 32-bit). As an example, the MMX instruction set provides instructions that can perform up to 8×8-bit operations in parallel. It can therefore be objected that the performance results presented in this work for the RDISK platform should be compared with software implementations that would take advantage of such architectural improvements, since they are very likely to significantly boost up software performance.

However these instructions were reported to provide only limited performance improvement [4], with speed-up only seldom above 2. In our case, as described in Section 8, this limited efficiency is worsen by the computation volume overhead induced by the SIMD execution model. For example, when using MMX instructions to compute 8 distance scores in parallel, the average number of useful iterations grows by a factor of almost 3, hence annihilating all the performance improvements due to parallelism.

7.3. Impact of Technology Improvement

As mentioned in Section 2, the RDISK prototype was designed using a low-cost FPGA of year 2001. However, FPGA technology is known to evolve very quickly with ever increasing density and prices dropping significantly every year. To understand the impact of this evolution, we considered an hypothetical implementation of RDISK using a 2004 FPGA (namely the Spartan-3 FPGA). For the same cost this new family offers five times the logic density of the current RDISK FPGA, with clock speed improvements in the range of 50%. According to our modeling, the op-

year	32-node RDISK	desktop PC	speedup
2001	~<100 s	30000 s(est.)	300
2004	~<10 s (est.)	15000 s	~1500

Table 1. Technological projections for FPGA versus CPU performance for a database of 30,000 images

timal implementation in this case is a 75-processor array operating at 66 MHz with a partitioning parameter $\sigma = 20$ (such a hardware filter can handle up to 1500 descriptors).

Following the same principle, we also considered a software implementation on a 2001 mid-end Intel processor (namely the Intel PIII-900 MHz). We used a very approximative estimate (50 % the performance of the P4). These estimates are summarized in Table 1. They show that when comparing 2001 technology (e.g. PIII against current RDISK) top 2004 technology (e.g. P4 against Spartan-3 based RDISK), we can observe that there is a growing gap between the software and the FPGA implementation performance in favor of the latter one. This suggests that an FPGA implementation of CBIR is likely to become more and more attractive compared with a software implementation.

8. Conclusion

In this paper we have proposed a parallel implementation for a Content Based Image Retrieval application on the RDISK cluster. This implementation combines the benefit of fine and coarse grain parallelism, with projected speedup up to 200. It is however to note that the scope of this work goes beyond a simple implementation work: our design approach followed a strict methodology that was validated on *real-life* data sets. This methodology allowed us to derive an *a priori* optimal hardware realization, by taking into account several system level parameters (resource usage, bandwidth, etc.). So far, we have restricted ourselves to images databases. However, we believe that other types of multimedia content-based search applications could benefit from a hardware acceleration on RDISK. We are currently investigating this direction.

References

[1] A. Acharya, M. Uysal, and J. H. Saltz. Active Disks: Programming Model, Algorithms and Evaluation. In *Architectural Support for Programming Languages and Operating Systems*, pages 81–91, 1998.

[2] L. Amsaleg and P. Gros. Content-Based Retrieval using Local Descriptors: Problems and Sssues from a Database Perspective. *Pattern Analysis and Applications*, 2001.

[3] E. Babb. Implementing a Relational Database by Means of Specialized Hardware. *ACM Transaction on Database Systems*, 4(1):1–29, 1979.

[4] R. Bhargava, L. K. John, B. L. Evans, and R. Radhakrishnan. Evaluating MMX Technology Using DSP and Multimedia Applications. In *Proceedings of the 31st annual ACM/IEEE international symposium on Microarchitecture*, 1998.

[5] R. Datta, J. Li, and J. Z. Wang. Content-Based Image Retrieval - Approaches and Trends of the New Age. In *International Workshop on Multimedia Information Retrieval*, 2005.

[6] S. Derrien, S. V. Rajopadhye, and S. Sur-Kolay. Combined Instruction and Loop Parallelism in Array Synthesis for FPGAs. In *ISSS'01 : Proceedings of the International Symposium on System Synthesis*, pages 165–170, 2001.

[7] C. Faloutsos. *Searching Multimedia Databases by Content*. Kluwer Academic Publishers, 1996.

[8] G. Govindu, L. Zhuo, S. Choi, and V. Prasanna. Analysis of High-performance Floating-point Arithmetic on FPGAs. In *Reconfigurable Architecture Workshop*, 2004.

[9] S. Guyetant, M. Giraud, L. L'Hours, S. Derrien, S. Rubini, D. Lavenier, and F. Raimbault. Cluster of Reconfigurable Nodes for Scanning Large Genomic Banks. *Parallel Computing*, 2005.

[10] K. Keeton, D. A. Patterson, and J. M. Hellerstein. A Case for Intelligent Disks (IDISKs). *SIGMOD Rec.*, 27(3):42–52, 1998.

[11] L. Kostoulas and I. Andreadis. Parallel Local Histogram Comparison Hardware Architecture for Content Based Image Retrieval. *Journal of Intelligent and Robotic Systems*, 2004.

[12] G. Memik, M. T. Kandemir, and A. Choudhary. Design and Evaluation of Smart Disk Cluster for DSS Commercial Workloads. *Journal of Parallel and Distributed Computing (JPDC)*, 61(11):1633–1664, 2001.

[13] D. Menard and O. Sentieys. Automatic Evaluation of the Accuracy of Fixed-Point Algorithms. In *Proceedings of Design, automation and test (DATE)*, 2002.

[14] K. Nakano and E. Takamichi. An Image Retrieval System Using FPGAs. In *Proceedings of ASPDAC*, 2003.

[15] F. A. P. Petitcolas, R. J. Anderson, and M. G. Kuhn. Attacks on Copyright Marking Systems. In *Proceedings of the Second International Workshop on Information Hiding*, 1998.

[16] P. Quinton. Automatic Synthesis of Systolic Arrays from Uniform Recurrent Equations. In *Internation Symposium on Computer Architecture*, pages 208–214, 1984.

[17] E. Riedel, C. Faloutsos, G. A. Gibson, and D. Nagle. Active Disks for Large-Scale Data Processing. *IEEE Computer*, june 2001.

[18] O. D. Robles, J. L. Bosque, L. Pastor, and A. 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.

[19] C. Skarpathiotis and K. Dimond. A Hardware Implementation of a Content Based Image Retrieval Algorithm. In *International Conference on Field Programmable Logic and Application (FPL)*, January 2004.