Architectural exploration of the H.264/AVC decoder onto a coarse-grain reconfigurable architecture

T. Cervero, A. Kanstein, S. López, B. De Sutter, R. Sarmiento and J.-Y. Mignolet

***Abstract*— The most recent video coding standard, H.264/AVC, imposes severe computational requirements in comparison with the previous ones. This fact makes necessary in order to overcome real-time constraints to count with efficient implementations in terms of performance and flexibility onto a proper architecture. For this purpose IMEC has developed a generic coarse-grained reconfigurable architecture named ADRES (*Architecture for Dynamically Reconfigurable Embedded Systems*) adapted to these exigencies. This paper presents some guides about the architectural exploration onto ADRES, which permit to point out to it as a good choice for mapping full multimedia applications, such as H.264/AVC decoders. The main goal of this paper is to present different ways to obtain performance improvements that directly depend of architectural modifications in ADRES, maximizing the performance of a baseline profile H.264/AVC decoder. In this sense, this work demonstrates that it is possible to improve performance results related to different parameters as well as to increase the degree of efficiency of ADRES resources.**

I. Introduction

An inherent desirable feature of modern multimedia devices is that they should be capable of bearing flexibility and performance requirements at the same time. The flexibility avoids creating new devices each time modifications are added into an application. In this sense, it is important to develop devices that support future improvements and modifications without modifying its structure. On the other hand, performance requirements involve parameters like speed, power consumption and silicon area. Reconfigurable architectures try to meet on a single device a degree of flexibility similar to CPUs while maintaining the ASICs levels of performance. Because reconfigurable processors execute efficiently only specific tasks, coupling them to a (soft) processor is imperative, with the subsequent communication drawback as a result. The ADRES (*Architecture for Dynamically Reconfigurable Embedded Systems*) architecture solves this problem with an ingenious solution, sharing resources between the reconfigurable array and the soft processor. The power efficiency, flexibility and high performance provided by reconfigurable architectures for the next generation of embedded multimedia devices point out ADRES, together with its compiler, as an ideal architectural choice for this purpose.

T. Cervero, S. López and R. Sarmiento are with the Institute for Applied Microelectronics (IUMA) and the Department of Electronic Engineering and Control (DIEA), University of Las Palmas de Gran Canaria, Spain. E-mail: {tcervero, seblopez, roberto}@iuma.ulpgc.es.

A. Kanstein is with Freescale, Inc., 134 Avenue due General Eisenhower, Toulouse, France. E-mail: A.Kanstein@freescale.com.

J.-Y. Mignolet and B. De Sutter are with the IMEC, Kapeldreef 75, Leuven, Belgium. E-mail: {i,mignolet@imec.be; bjorn.desutter@elis.ugent.be}

In addition, ADRES opens a wide range of architectural combinations thanks to its generic template. However, these characteristics generate an enormous design space that makes it difficult to find optimized architectures.

Within the multimedia applications, video coding standards have been rapidly improved and developed during the last decade. One of the most recent video standard is the H.264/AVC [1], which incorporates several benefits with respect to its antecessors. This standard gains in flexibility into the coding/decoding process, and as a direct consequence, it gets to reduce the transmission rates a 50% and 35% in comparison with MPEG-2 [2] and MPEG-4 [3], respectively. However, the performance improvement comes with an associated increase in the resulting implementation complexity. Due to this reason, the previous idea of meeting flexibility and performance over the same device is even more critical.

In order to cope with these challenging requirements, we believe that the most appropriate device to map an application like the H.264/AVC decoder comes from the coarse grained architectural group [4]. Inside this extensive group of alternatives, IMEC's ADRES architecture shapes as an adequate alternative. At the moment, some architectural explorations [13], [14], [15] have been done related with simple kernels (FFT and IDCT), but no one has studied the impact over the performance of mapping a complete application, such as H.264/AVC baseline video decoder, onto ADRES.

This paper presents the results of an architectural exploration study in which we have mapped a baseline profile H.264/AVC decoder onto several ADRES architectures. Its main contribution is to present more insights into the usefulness of different architectural features and parameters.

The remainder of this paper is organized as follows. Section II introduces the H.264/AVC decoder structure and its functionality. Section III describes the main characteristics of ADRES and its associated programming tool, the DRESC compiler. Moreover, Section IV explains the architectural exploration done in order to find an appropriate template adequate to the H.264/AVC decoder necessities. Finally, Section V exposes a set of general conclusions about the obtained results and future research lines.

II. The H.264/AVC decoder

As it is shown in Figure 1, the H.264/AVC decoder is separated into different functional blocks, each one with an specific task associated.

The input bitstream is loaded into a memory buffer, with the objective of being parsed and decoded by the entropy decoder block. The syntax elements obtained after this

process for each macroblock (16×16 luminance pixels and two blocks of 8×8 chrominance pixels) are demultiplexed and sent to the different functional kernels involved in the decoding process. In particular, the syntax elements related with the coding of the luminance and chrominance residual samples of the current macroblock (MB) are handled by the upper branch, showed in Figure 1, composed by the inverse quantization and the inverse transform.

In parallel, the predictor finds the macroblock of reference using the temporal (Motion compensation block) or spatial (Intra prediction block) redundancy found in the previously decoded data, according to the information received in the MB layer.

Adding this information (quantified and transformed residual samples plus the MB of reference), the original MB is recovered. Finally, before sending this MB as the output of the system, the MB is filtered by the deblocking filter block, in order to reduce annoying blocking artifacts.

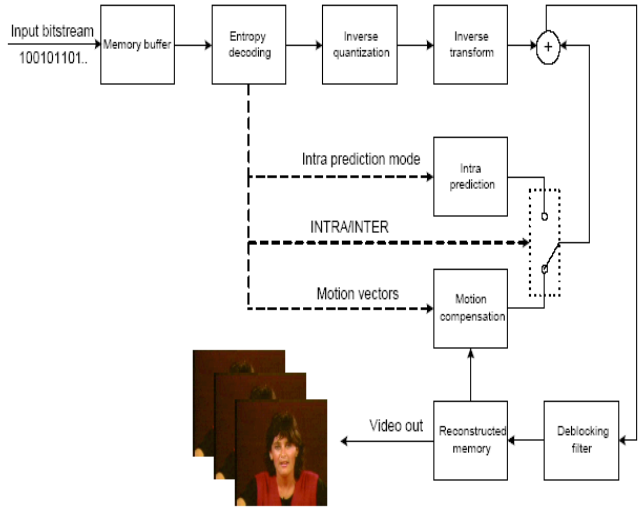

Fig. 1. Block diagram of a generic H.264/AVC decoder

It is important to mention that the selected baseline H.264/AVC decoder used in this research is the ANSI C implementation from the *libavcodec* open-source library [5]. In this implementation, the most complex blocks are the motion compensation and the deblocking filter due to high quantity of data and memory accesses that they need. The software mapping effort took 3-4 man months, and it involved adapting the memory consumption to the scratch-pad based memory hierarchy of our prototype ADRES implementation, and the restyling of inner loops to fit our the ADRES architecture better, i.e. to make them better software-pipelining candidates. This limited effort that was required, starting from an open-source software implementation, illustrates that indeed ADRES provides a very flexible solution.

III. ADRES/DRESC FRAMEWORK

A. ADRES architecture

The ADRES architecture is a flexible template consisting of a coarse grain reconfigurable array (CGRA) and a very long instruction word (VLIW) processor within the same physical entity [6], [7]. The CGRA has been designed to accelerate the intensive computational operations of application kernels by exploiting loop level parallelism (LLP), whereas the remaining code is mapped onto the VLIW processor that exploits instruction level parallelism (ILP).

This design overcomes the common drawback associated with CGRAs, which usually duplicate storage resources to transfer data between the CGRA and the processor, by sharing the central register files and memory interfaces between the array and the VLIW processor. This also circumvents the need to spend execution time on data transfers. Figure 2 depicts a general scheme of ADRES where the CGRA consists of an array of functional units (FUs) and of small register files (RFs). The VLIW processor is mainly formed by a subset of the FUs and by the global data register file (GDRF), that it shared with the CGRA. Since the VLIW mode and the CGRA mode operate exclusively, i.e. one of the two is executing at any point in time, there is no contention for resources between both modes.

The hardware design flexibility of ADRES is concentrated in its CGRA, for which the designer can choose the number of FUs, the operations supported by the FUs, whether or not local register files are shared by multiple FUs, how many ports they have, what the data routing resources are (buses, multiplexor, wires), etc. Obviously this design flexibility also results in an optimization problem: how do I design a specific ADRES architecture that is optimal for some application domain with respect to performance or energy, or both. Fortunately, the whole ADRES tool chain, including the ANSI C compiler and the VHDL generator, is retargetable to any ADRES instance by means of an instance specification in the XML format.

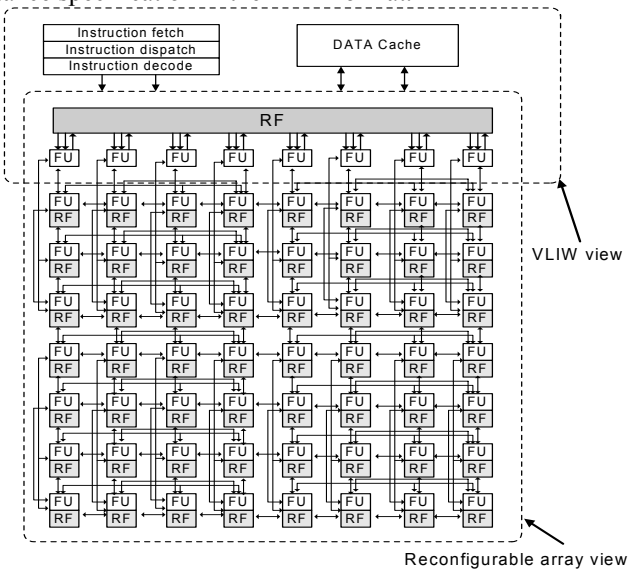

Fig. 2: Architecture of the ADRES Coarse-Grain Reconfigurable Array

Figure 3 shows the scheme of the basic processing element in ADRES, also depicting its detailed data path. This kind of FU performs coarse-grained operations on 16, 32 or 64 bits of data. Moreover, it makes use of predicated signals (*pred* and *pred_dst1/ pred_dst2* ports) with the main goal of transforming the control flow inside loops into conditional execution operations.

With the help of local RFs inside the CGRA, it is possible to reduce the number of accesses to the global memory and the global data register file, and also to accelerate the data transfers. This happens because local RFs are smaller and

faster than the shared RFs, and usually they have less ports than the global RFs. Thus, they also consume less power.

The output buffers at each FU output ensure a good timing by registering each output, while the multiplexers have the ability to route data from different sources. As FUs are reconfigurable elements, each one has associated a configuration RAM that acts as an instruction memory to control the functionality of the component. It stores a number of configuration contexts locally, which are loaded on a cycle-by-cycle basis by the CGRA controller as are the addresses that are set at the register ports, and the selector bits of all the muxes.

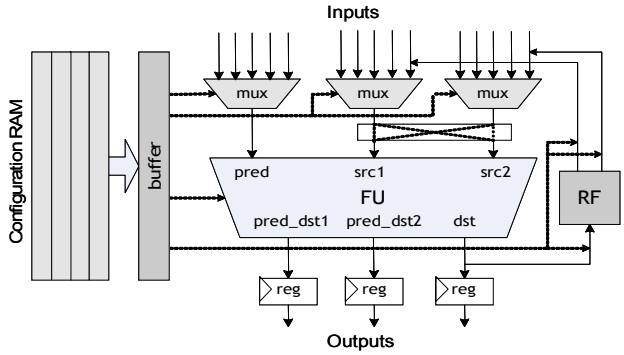

Fig. 3: ADRES Coarse-Grain Array Node

Finally, the interconnections between the different types of resources in ADRES can be chosen according to the needs of an application domain. Common topologies are the *mesh* (where all direct neighbors are connected) or the *meshplus* (which is a extended mesh topology that include connections with FUs situated in a distance of one FU) [8].

The operation mode of an ADRES is as follows: non-loop code is executed on the VLIW processor, which can call a loop just like it can call a subroutine. Control is then transferred to the CGRA unit, which executes a loop, and control is transferred back to the VLIW mode when a loop exit condition is triggered.

B. DRESC framework

To program ADRES processors, IMEC has developed a compiler framework. This framework includes DRESC (*Dynamically Reconfigurable Embedded System Compiler*), a C compiler perfectly adapted to ADRES. It maps computation-intensive kernels, typically dataflow loops, onto the reconfigurable array, while the rest of the code is mapped onto the VLIW processor. One main advantage of the DRESC framework is its flexibility, as the tools are designed to be retargetable within the ADRES template by using the XML architecture description to retarget the compilation of software on the fly.

The full compiler framework is depicted in Figure 4. The process starts from a C-description of the application. The profiling step identifies kernels to be mapped onto the CGRA at the execution time. Manual source-level transformations are needed to bring hot loops in a good shape to maximize performance. To the best of our knowledge, such C-code messaging is needed with every tool chain. The next step is carried out by IMPACT [9], the front-end compiler that we use for high-level code optimizations. Its output is an ADRES-instance-independent intermediate representation called *Lcode* which is used together with the XML-based architecture as an input to the DRESC compiler, which retargets to an ADRES instance automatically.

The DRESC compiler generates both CGRA and VLIW code. For the former, a modulo scheduling algorithm is applied in order to exploit the highest possible level of loop-level parallelism, whereas the VLIW code is generated by meant of a traditional ILP scheduling technique.

The modulo scheduling algorithm utilizes a graph-based architecture representation, called MRRG (Modulo Routing Resource Graph) [10], to model resources in a unified way, expose routing possibilities and enforce modulo constraints. The algorithm is based on congestion negotiation and simulated annealing methods. Starting from an invalid schedule that overuses resources, it tries to reduce the overuse along the time until finding a valid schedule.

The DRESC compiler automatically identifies and handles communications between the two ADRES parts. This is

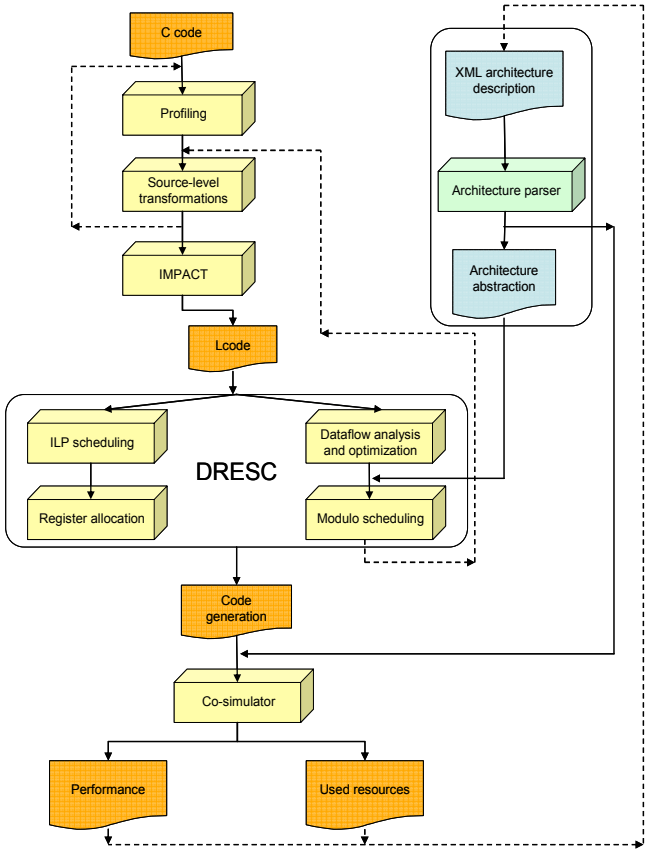

Fig. 4: Dynamically Reconfigurable Embedded System Compiler (DRESC) Framework

achieved by ensuring that there is a match between the register allocations of both modes for the shared register files. This way, the VLIW automatically puts the data to be consumed by the loops in the place expected by the loops.

Finally, the tools also generate a co-simulator, using the architectural description and the scheduled code as inputs. This co-simulator verifies the execution results on the target ADRES architecture and collects many statistics such as the total number of instructions and the total number of cycles. With this output information, it is possible to modify the XML-based architecture with the objective of adapting the ADRES resources to the application features.

IV. ADRES ARCHITECTURAL EXPLORATION

While software improvements are done manually or automatically (using in the first case manual code transformations, and in the second case both IMPACT and DRESC compilers insight the design flow), hardware enhancements require to overcome more trade-offs.

The trade-offs involved in choosing the optimal architectural parameters are not obvious. This results from the fact that there is a large design space, with many feature choices that depend on each other in ways which are not obvious, and from the fact that the compiler's simulated-annealing makes it difficult to correlate program properties to architectural features. To offer more insights in the usefulness of certain design features for the domain of multimedia and of H.264/AVC decoding in particular, this paper shows the effects of varying important architectural aspects such as interconnect topology, FUs, memory ports, and distributed RFs. While the results cannot produce an optimal architecture instance, they point to interesting architectural design choices, while offering a better understanding of the effects of different parameter choices and their interaction over the final application benefits.

A. *Preliminary analysis*

As it was mentioned before, ADRES structure has two operation modes. The soft processor, tries to exploit the instruction level parallelism (ILP), while the CGRA is prepared to exploit the loop level parallelism (LLP). In this sense, it is important to analyze the H.264/AVC baseline video decoder code for detecting which functions are good candidates to be executed onto the CGRA operation.

The profiling of the code, which was done before the architectural exploration, shows that there are several functions with a high computational cost. A significant problem is the amount of sequential code and the complexity of the computations; as there are many kernels with a lot of control structures, which makes the mapping onto the CGRA more difficult.

Regarding to the high computational cost of many kernels within the H.264/AVC decoder, such as Motion Compensation (MC), Deblocking Filter (DF), Inverse Direct Cosine Transform (IDCT), Control Frame and Memory, Table 1 represents numerically the impact of mapping the application onto both systems: a VLIW processor (VLIW column) and an ADRES architecture (VLIW+CGRA column). The selected measurement parameter has been the number of execution cycles, since it is a clear and fast indicator.

Functional blocks	VLIW	Usage (%)	VLIW + CGRA	Usage (%)
DF	74.250.346	32,04%	21.632.518	17,45%
MC	40.596.396	17,52%	29.383.548	23,71%
IDCT	22.060.046	9,52%	6.588.902	5,32%
Control Frame	6.907.365	2,98%	3.234.561	2,61%
Memory	6.821.508	2,93%	2.391.550	1,93%
Previous Blocks	150.637.861	65,01%	63.231.079	51,02%
Other Blocks	81.083.634	34,99%	60.696.677	48,98%
Full system	231.721.495	100%	123.927.756	100%

Table 1 Comparison between VLIW processor and ADRES execution

Column called *Usage (%)* on Table 1 represents the percentage assigned to each functional block within the application execution. In both cases, DF and MC kernels reach higher values of computational cost within the whole execution process. In the first case, (when VLIW processor is running alone) DF and MC kernels suppose the 49,56% of the total execution time, while in the second case (ADRES execution) this value is decreased till 41,16%.

In spite of *Motion Compensation* block is mainly formed by simple loops, they are executed a lot of times, which results in a long computation time (17,52% of the total computational cost onto a VLIW processor). When this block is executed onto ADRES, the parallelization makes possible to reduce the number of execution cycles up to the 50%.

Deblocking Filter (DF), Inverse Discrete Cosine Transform (IDCT), Control frame and Memory kernels, are composed by nested loops, a lot of control flow and data dependencies. Within these kernels, DF is the most remarkable, since it needs the higher number of cycles for execution (32,04% of the total number of cycles running on a VLIW processor) Hence, the main goal of mapping these kernels onto the CGRA is to overcome these difficulties and to raise better results, making them more friendliness. In these cases, when they are executed onto ADRES architectures, their number of cycles is reduced. Attending to the results on Table 1, it is obvious that ADRES is faster (VLIW+CGRA column), since it executes the same number of operations that VLIW processor, but using lower numbers of cycles. This is due to the ADRES property that allows parallelization.

The architectural exploration can be partitioned into four general groups of architectural modifications: (1) operations distribution, (2) interconnect topology, (3) communications between the reconfigurable array and the VLIW processor, and (4) dimensions of the array. The first analyzes the impact of the distribution of the operations through the heterogeneous FUs. While each FU is capable of executing basic operations such as arithmetic, logic, shift, and predicate operations, multipliers and load/store units are too expensive to be present in each FU. Due to this reason, a first type of design choice involves the number of such units, and their distribution along the heterogeneous array. This choice depends on the occurrence of multiplication operations in the application loops and on the ratio between load/store operations and actual computations.

The second group of modifications within the architectural exploration is related to the previous one. In this case, the challenge is to minimize the number of wires and busses in the interconnection network (topology) in order to minimize the cost of transferring data through the array. Although avoiding the transfer of data through FUs, in which case they cannot perform computations on the data, there should still be an appropriate number of connections. These modifications to an architecture's interconnect are mainly steered by the data flow graph shapes observed in the application loops.

The third group of architectural modifications relates to the shared resources of the VLIW and the CGRA. For example, local data RFs are beneficial into the CGRA operation because they avoid accesses to the GDRF and they create additional routing for the DRESC compiler. However, it is interesting to decrease their number because of their elevated area occupancy. Another critical shared resource is the GDRF, being its impact on the performance higher than the local RFs [13].

Finally, the fourth group of modifications we have studied relates to the size of the array, i.e. to the number of FUs and RFs in it. Basically, this design option relates to the amount of available loop-level parallelism in the applications. In video codecs, this parallelism is largely limited by the fact that many inner loops operate on MBs of 4x4 pixels only. As a consequence, the loop-level parallelism is rather limited. The final performance obtained with each architectural template depends on its hardware characteristics and also on the behaviour of the loops executed on it. Obviously, any good design point will be optimized for the combination of loops to be executed on the array, rather than for any particular loop.

B. *Architectural exploration*

This section presents the most remarkable results of the exploration. For this task, we have started from an ADRES multimedia instance that was build by an IMEC team within of a previous exploration done with simple kernels [15], as part of a demonstration prototype. This prototype featured rather small L1 scratch-pad memories (32KB) and a limited number of FUs and RFs because they had to be implemented on a limited-size FPGA. Because of the small memories, the source code we used only operates on CIF (352x288 pixels) and QCIF (176x144 pixels) video formats. Experiments with other implementations show that the results extrapolate well to larger frame sizes, so we do not consider this as a fundamental limitation.

In this paper, we present the performance results for four architectures, summarized in Table 2. First, there is a 4x4 architecture, named *Reference*. The main value of this version is to serve as a reference point, as it was used to validate the correctness of our whole tool flow with IDCT and FFT kernels. Moreover, this version has been already used in many research experiments [11], [12], [15].

Secondly, there is a 3x4 architecture which basically reduces the size of the 4x4 *Reference*. This is a version that permits to meet easily the restrictions imposed by the FPGA size constraints.

The third architecture, named 6RF architecture, is a 4x4 architecture that includes 6 shared local RFs (shared between two FUs) rather than 12 non-shared ones. Thus, the area of the core is reduced without reducing the potential maximum throughput.

Finally, a much larger core is included in the results, featuring 4x8 FUs and 24 local, shared RFs.

All of the architectures presented in this section feature a meshplus interconnect between FUs. Although this paper only presents results for the four most interesting ADRES instances, other probes during our experiments demonstrated that architectures with sizes smaller than 3x4, e.g. 2x2, are not large enough to map the H.264/AVC decoder properly.

Characteristics	Architectures			
	Reference	3x4	6RF	4x8
Nº of CGRA FUs	16 4 rows 4 cols.	12 3 rows 4 cols.	16 4 rows 4 cols.	32 4 rows 8 cols.
VLIW issue width	3	3	3	4
Nº of local RFs	12	8	6	24
Nº of MUL	6	5	6	11
Nº of LD/ST	4	4	4	8

Table 2: Architectures definition

All these architectures have had a more than acceptable behaviour after executing the H.264/AVC baseline decoder along the complete architectural exploration. Next section shows their final global results with the pair of input video sequences considered in this work, football.qcif and football.cif.

C. *Results*

Tables 3 and 4 depict the most relevant results obtained onto the CGRA, after simulating the H.264/AVC baseline decoder with four ADRES templates explained in the previous section.

Blocks	N Functions	Mapped loops	Average IPC
Motion Compensation			
3x4	9	12/12	7,487
Reference			11,381
6RF			11,167
4x8			16,455
Deblocking Filter			
3x4	1	6/6	9,759
Reference			11,164
6RF			10,411
4x8			14,569
IDCT			
3x4	2	5/5	9,428
Reference			10,889
6RF			10,305
4x8			15,444
Control			
3x4	3	4/4	5,758
Reference			6,336
6RF			4,602
4x8			8,480
Memory			
3x4	2	5/5	3,25
Reference			3,25
6RF			1,875
4x8			3,25

Table 3: football.QCIF performance results with ADRES

IPC is a common parameter which indicates the number of instruction per cycles executed. In this paper, *Average IPC* has been selected because it avoids speculations about the performance and reflects the general behaviour of the system within the CGRA, without influences of the external variations or dependencies. This parameter is calculated as an average of the total IPCs involved in the CGRA process.

Blocks	N Functions	mapped loops	Average IPC
Motion Compensation			
3x4	9	12/12	9,146
Reference			11,417
6RF			10,702
4x8			16,455
Deblocking Filter			
3x4	1	6/6	9,187
Reference			10,905
6RF			10,411
4x8			13,717
IDCT			
3x4	2	5/5	9,428
Reference			10,889
6RF			9,778
4x8			15,444
Control			
3x4	3	4/4	5,758
Reference			6,336
6RF			4,601
4x8			8,647
Memory			
3x4	2	5/5	3,125
Reference		5/5	3,125
6RF		4/5	1,875
4x8		5/5	3,125

Table 4:-football.CIF performance results with ADRES

In both tables, the three first functional blocks (MC, DF and IDCT) follow the same pattern, that is, the biggest architecture obtains the highest IPC. However, regarding to Control and Memory functional blocks the order varies, being the 6RF the worst one.

These data are coherent, because the biggest architecture can use more quantity of resources at the same time. A remarkable result is the fact that there is not a big difference among the CGRA behaviour when the image format is QCIF or CIF, despite of the increment of computation between both. The average IPC remains quite similar in both tables for all cases. Furthermore, this parameter allows computing the *density* of the array.

Density is the relationship between the IPC and the number of FUs into the architecture, and it gives idea about the utilization of the FUs. Tables 5 and 6 depict the percentage of densities calculated through the average IPC parameter values.

QCIF (%)	MC	DF	IDCT	CONTROL	MEMORY
3X4	60,52%	77,89%	78,5%	45,88%	25,83%
Reference	64,78%	69,78%	67,73%	51,97%	19,37%
6RF	62,09%	65,09%	61,46%	28,99%	13,12%
4X8	45,12%	45,53%	48,54%	24,87%	9,69%

Table 5: Density with a football.qcif image

CIF (%)	MC	DF	IDCT	CONTROL	MEMORY
3X4	70,88%	77,07%	94,33%	45,88%	25,83%
Reference	64,45%	68,16%	69,17%	37,12%	19,37%
6RF	60,7%	65,07%	61,46%	29%	13,12%
4X8	45,12%	42,87%	48,54%	23,98%	9,68%

Table 6: Density with a football.cif image

These tables, 5 and 6, show that the 4x8 architecture has too many idle FUs. These results about the 4x8 architecture are due to the decoder implementation, not to the compiler limitations. However, although this template does not fully exploit its resources, it is faster that the others on execution time.

Finally it is important to remark that an inconvenient associated with this architectural exploration is that it does not permit to know a priori anything about the area and power consumption. However, it is possible to realize a simple subjective analysis in order to establish some comparisons at this respect. Hence, the couple of physical parameters, power consumption and area, should be analyzed with a set of appropriate tools, in order to consider all the factors involved [14] into a real implementation.

V. Conclusions and Future Work

This paper presents an architectural exploration over the ADRES architecture when the H.264/AVC decoder is executed.

This work has demonstrated that using an ADRES architecture to implement applications with a high level of computation is an appropriate decision, and it is possible to enhance the performance results. Of course, the characteristics of the final selected architecture template depend directly on which architectural parameter is the most important.

Further research can be oriented to develop structures with multiple architectures where each one can execute specific tasks in hardware (mapping the most computational intensive operations into the CGRA) and sharing the software execution in an unique VLIW or in multiple processors. Additionally, future works can enforce the study and analysis on silicon area and power consumption with different ADRES templates.

Acknowledgment

The authors wish to thank IMEC for supporting the student internships which made this work possible, and for providing access to the tools.

References

[1] Wiegand, T. (Ed.), "Draft ITU-T Recommendation H.264 and Draft ISO/IEC 14496-10 AVC," *Joint Video Team of ISO/IEC/JTC1/SC29/WG11 & ITU-T SG16/Q.6 Doc. JVT-G050*, Pattaya, Thailand, March 2003.

[2] ISO/IEC JTC1/SC29 CD 11544, "*Coded Representation of Picture and Audio Information - Progressive Bi-Level Image Compression*", Recommendation H.262 November, 1993.

[3] MPEG-4 Part 2: Visual (IS 14496-2), Doc. N2502, Atlantic City, N.J., USA, October 1998.

[4] Hartenstein, R. *"Coarse Grain Reconfigurable Architectures*", Design Automation Conference, pp.564-569. Las Vegas (USA), June 2001.

[5] Available at http://sourceforge.net/projects/ffmpeg [Online]

[6] Mei, B. "*A Coarse-Grained Reconfigurable Architecture Template and its Compilation Techniques"*, PhD Thesis IMEC, January 2005.

[7] Mei, B. "*Adres: An Architecture with Tightly Coupled VLIW Processor and Coarse-Grained Reconfigurable Matrix*". Proc. Field-Programmable Logic and Applications, pp. 61-70, Springer, 2003.

[8] Bouwens, F. "*Power and Performance Optimization for ADRES: Analysis, Optimization and Synthesis of a Coarse-Grained Reconfigurable* Array*"* MSc. Thesis in Computer Engineering, IMEC (Belgium) & Delft University of Technology (The Netherland), 2006.

[9] Chang, P.P.; Mahlke, S. A.; Chen, W. Y.; Warter, N. J.; Hwu, W. W. "*IMPACT: An Architectural Framework for Multiple-Instruction-Issue Processors*", *Proc. of the 18th International Symposium on Computer Architecture (ISCA)*, pp. 266-275, 1991

[10] Quan, X.; Shijie, J.; Jiandong, Z. "*H.264/AVC Baseline Profile Decoder Optimizations on Independent Platform*", *International Conference on Wireless Communications, Networking and Mobile Computing,* Volume 2, pp. 1253 - 1256, September, 2005

[11] Hernández, H.; Kanstein, A.; López, S.; López, J.F.; Beredovic, M.; Sarmiento, R. and Mignolet, J.-Y "*Mapping of the H.264/AVC motion compensation algorithm onto a coarse-grain reconfigurable array*", Conference on Design of Circuits and Integrated Systems, DCIS, pp. .Barcelona (Spain), November 2006.

[12] Arbelo, C.; Kanstein, A.; Lopez, S.; Lopez J.F.; Beredovic, M.; Sarmiento, R. and Mignolet, J.-Y; "*Mapping Control-Intensive Video Kernels onto a Coarse-Grained Reconfigurable Array: the H.264/AVC Deblocking Filter*", The European Event for Electronic System Design & Test, DATE, pp.177-182. Acropolis, Nice (France), April 2007.

[13] Kwok,Z.; Wilton, S.J."*Register File architecture optimization in a coarse-grained reconfigurable architecture*" 13th annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), vol. 00, pp. 1-10, 2005

[14] Bouwens, F.; Berekovic, M.; Kanstein, A.; Gaydadjiev, G. "*Architectural exploration of the ADRES coarse-grained reconfigurable array*" ARC 2007, LNCS 4419 pp. 1-13, Springer.

[15] Bouwens, F.; Berekovic, M.; De Sutter, B.; Gaydadjiev, G. "*Architecture enhancements for the ADRES coarse-grained reconfigurable array*" HiPEAC 2008, LNCS 4917, pp. 66-81, Springer