Still Image Processing on Coarse-Grained Reconfigurable Array Architectures

Matthias Hartmann[1] Vassilis Pantazis[2] Tom Vander Aa[2] Mladen Berekovic[2]
Christian Hochberger[3] Bjorn de Sutter[2]

***Abstract*—Due to the increasing demands on efficiency, performance and flexibility reconfigurable computational architectures are very promising candidates in embedded systems design. Recently coarse-grained reconfigurable array architectures (CGRAs), such as the ADRES CGRA and its corresponding DRESC compiler are gaining more popularity due to several technological breakthroughs in this area. We investigate the mapping of two image processing algorithms, Wavelet encoding and decoding, and TIFF compression on this novel type of array architectures in a systematic way.**

The results of our experiments show that CGRAs based on ADRES and its DRESC compiler technology deliver improved performance levels for these two benchmark applications when compared to results obtained on a state-of-the art commercial DSP platform, the c64x DSP from Texas Instruments. ADRES/DRESC can beat its performance by at least 50% in cycle count and the power consumption even drops to 10% of the published numbers of the c64x DSP.

I. Introduction

A new branch of programmable processor architectures for demanding DSP applications has emerged in the recent years, such as image processing or video coding: coarse-grained reconfigurable architectures (CGRAs). Even though many CGRAs were proposed before ([4], [13], [8], [1]) none of them have yet been broadly used, some due to the difficult programming models and others due to their extensive overuse of resources compared to other DSP processors. The novel ADRES (Architecture for Dynamically Reconfigurable Embedded Systems) CGRA addresses a number of these insufficiencies and its corresponding compiler DRESC (Dynamically Reconfigurable Embedded System Compiler) compiles C code with a very high utilization (50% to 80%). Furthermore it was shown that the ADRES and its DRESC is highly suitable for video decoding [2] and for wireless applications [12]. This paper will show its additional suitability for still image processing.

The ADRES architecture closely connects a very-long instruction word (VLIW) processor and a coarse-grained array by providing two functional views of the same physical resources. The VLIW part enables the mapping of complex, control-flow intensive applications, which is missing in other published CGRA implementations. The array part offers unprecedented loop accelerations. An easy mapping of applications written in C onto the VLIW and the array mode is ensured by the DRESC compiler framework. A shared register file between these two modes, which also functions as storage for live-in and live-out variables for the loop mode, provides a minimum of communication and mode-switching costs and makes it possible to generate code for both modes, including the data transfer operations. Finally, ADRES is a template instead of a concrete processor architecture and with the retargetable compilation support from DRESC, architectural exploration becomes possible to discover better architecture instances or design domain-specific architectures.

This paper presents how two image processing applications, a wavelet transform [14] and the Tiff2BW benchmark from the MiBench benchmark suite [6] were mapped on multiple ADRES instances. For these applications, being applied in still cameras as part of ever more complex image processing algorithms, high performance and low-power consumption are also becoming increasingly important. We investigate if the ADRES architecture and its DRESC compiler are suitable for image processing applications. To optimize the performance and power consumption, we explorated the available loop level parallelism, and explored multiple ADRES architectures During this exploration, both applications were first mapped to the VLIW mode only and later the loops were mapped onto the array mode. The applied optimizations, the results and power numbers are discussed in this paper and compared to the state-of-the-art VLIW-DSP processor c64x from TI [7].

The rest of this paper is organized as follows. Section 2 introduces the ADRES architecture template and its corresponding DRESC compiler. Section 3 presents the Tiff2BW benchmark and its optimization for ADRES and DRESC. Section 4 does the same for the wavelet transformation. In Section 5 we show the obtained results on a multitude of different ADRES instances and compare them with the DSP processor c64x from TI.

II. ADRES CGRA Architecture and DRESC Compiler

The ADRES architecture template [10], as shown in Figure 1, consists of an array of basic components, including functional units (FUs), register files (RFs) and and interconnect network. The top row can act as a tightly coupled VLIW processor in the same physical entity. The array contains three types of basic components: FUs, storages resources such as RFs and read-only data memoriess, and interconnects that include wires, muxes and busses. The ADRES architecture is a flexible template that can be freely specified by an XML-based architecture specification language as an arbitrary combination of those elements. The two parts of ADRES share the shared register file and

[1] Department of Electrical Engineering, University of Technology Dresden, Helmholtzstrasse 18, 01062 Dresden, Germany
[2] IMEC Leuven, Kapeldreef 75, B-3001 Leuven, Belgium
[3] Department of Computer Science, University of Technology Dresden, Helmholtzstrasse 10, 01062 Dresden, Germany

load/store units. The computation-intensive kernels, typically dataflow loops, are mapped onto the reconfigurable array by the compiler using a modulo scheduling technique to implement software pipelining and to exploit the highest possible parallelism, whereas the remaining code is mapped onto the VLIW processor. The data communication between the VLIW processor and the reconfigurable array is performed through the shared RF and shared memory.

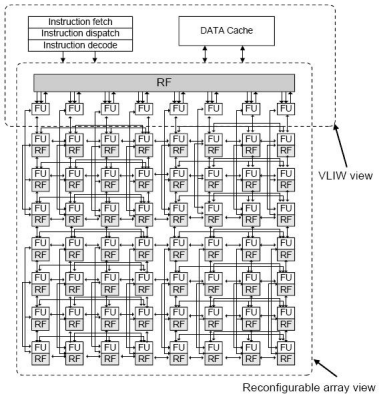

Fig. 1. Architecture of the ADRES reconfigurable array

In contrast to FPGAs, the FU in ADRES performs coarse-grained operations on 32 bits of data, e.g. ADD, MUL, and Shift. To remove the control flow inside loops, the FUs support predicated operations for conditional execution. High operation frequencies can be obtained by inserting pipelining latches at the outputs of FUs. The values produced by an FU can be written to a local RF, which is usually small and has fewer ports than the shared RF, or routed directly to the inputs of other FUs. The multiplexers are used for routing data from different sources. The configuration RAM acts as a (VLIW-) instruction memory to control these components. It stores a number of configuration contexts locally, which are loaded on a cycle-by-cycle basis. As such, the ADRES CGRA is a dynamically reconfigurable architecture.

The DRESC compiler tool chain consists of the IMPACT C compiler frontend [5] and of the DRESC compiler backend. IMPACT, a VLIW compiler framework, profiles and parses the C source code to an intermediate representation (Lcode), and applies several optimizations. These include extensive inlining and hyperblock formation by means of predication to eliminate control flow from inner loops. Those loops are then mapped onto the ADRES array mode with a modulo-scheduling algorithm [11] exploiting the high parallelism of the loop kernels. This array mapping is fully retargetable, as the target ADRES instance is desribed in an XML file that is fed to the compiler together with the C code of the application. Traditional ILP scheduling techniques are applied to achieve high performance in the non-kernel parts of the application by executing them on the VLIW mode. The DRESC compiler backend generates scheduled code for both the CGRA and the VLIW, which can be simulated by a co-simulator.

A. Experimental Setup

As mentioned in the previous section the DRESC compiler supports not just one specific architecure, but a wide range of ADRES architecture instances, which can be defined by an XML-based description. The experiments described in this paper were carried out on different instances with different numbers of FUs, different word width, different setup for the local RFs, and different interconnects between the FUs. More specific details about those architectures are shown in Table I.

III. Tiff2BW Benchmark

The original source code of the Tiff2BW benchmark was taken from the MiBench Benchmark Suite [6]. It is part of a set of benchmarks that tries to evaluate embedded devices for consumer applications. The benchmark converts a coloured TIFF image into a black and white TIFF image while decreasing the size of its file by the factor of 3. In order to make the results comparable the benchmark also includes two standard images of which one is transformed. The large TIFF image has a dimension of 3069 by 3100 pixels with 24 bits per pixel. Therefore the size of this image is 27.2 MB. The second in the benchmark included TIFF image has a size of 6.5 MB and a dimension of 1520 by 1496 pixels. Since the size itself is linearly influencing the number of cycles spent processing the TIFF image, the smaller TIFF image was used for the ADRES simulations in order to reduce the simulation time.

A. Transformation

In the Tiff2BW benchmark the transformation to a black and white image consists of one loop, whose source code is shown in Figure 2. This loop reads three values from an input buffer, representing the values for the red, green and blue component of the pixel, and weights them with three different but constant values. Those weighted values will be summed up and stored in an output buffer. Figure 2 also shows the resulting dataflow graph.

```
while (n-- > 0)
{
  v = red*(*rgb++);
  v += green*(*rgb++);
  v += blue*(*rgb++);
  *out++ = v>>8;
}
```

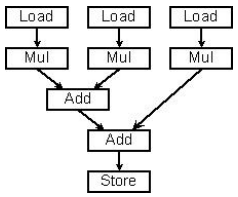

Fig. 2. Source code & dataflow graph of the transformation in the Tiff2BW benchmark

B. Optimization

In order to achieve optimal performance on a given ADRES architecture, a number of source-level transformations usually need to be applied that expose more parallelism to the compiler. All of those transformations are done on the C source code level. For the applications presented in this paper, all applied source level transformations were evaluated on the 4x4 multimedia ADRES instance in order to ensure that they effectively result in faster execution. Table II shows the simulation results of

Template	Word Width	Number of FUs	Number of FUs with Load/Store capability	Number of RFs	Connections
4x4 multimedia	32	16	4	12	nearest neighbor
4x4 wireless	64	16	4	13	mesh-plus routing
2x2 alternative [3]	32	4	2	1 for 4 words	morphosys routing
4x4_all alternative [3]	32	16	4	12, connected to each FU	mesh-plus routing
4x4_4L alternative [3]	32	16	4	4, each for 4 words	morphosys routing
4x4_16L alternative [3]	32	16	4	4, each for 64 words	morphosys routing
8x8 alternative [3]	32	64	8	16, each for 4 words	morphosys routing

TABLE I

Different ADRES architectures and their characteristics

the original source as well as of some of the transformed versions, which are discussed in the following paragraphs.

As a first optimization step, induction variable strength reduction was applied to the loop to eliminate the live-in variable n. While the simulation results in Table II show that the cycle count is not influenced significantly by this optimization, the number of executed instructions drops significantly. This prepares the loop for more important optimizations such as loop unrolling.

After removing the induction variable n, the loop was unrolled several times. Since there is no data dependency between different iterations of the loop, the unrolled loop has no additional restrictions that follow from data dependencies. Finding effective schedules, with higher numbers of instructions per cycle (IPC) executed is therefore made easier for the DRESC compiler, as it now has more operations to schedule. Four versions of differently unrolled loops are shown in Table II.

Experiments with the unrolled loops have shown that the improvements in performance decrease if a loop is unrolled too much. For the 4x4 multimedia ADRES instance, the results in Table II indicate that unrolling should be limited to seven times. The upper bound on loop unrolling follows from the fact that the unrolled loops become too big for the compiler too explore a lot of the scheduling space. This is also apparant from the increasing compilation times of the loops, that follow from the fact that the modulo schedule algorithm used in DRESC relies on simulated-annealing to explore the scheduling space without being trapped in suboptimal schedules. Research on possible improvements of the compiler for large dataflow graphs is future work.

The performance result of all versions are also shown in Table II. It can be seen that the number of cycles needed to perform the conversion was halved compared to the original source code, and the total number of instructions executed was decreased by 33%. The final results for a wider range of ADRES architecture are shown in the Section V.

IV. Wavelet Transformation

Wavelet-based coding is a powerful enabling technology for scalable applications. It is based on the principles of the Wavelet Transform, which is a special case of a sub band transform producing a type of localized time-frequency analysis. The wavelet transform implementation used in these experiments is very similar to the one used in the JPEG2000 still image compression standard [14].

A. Transformation

The wavelet transform application we used applies transformations in two phases. First, in a forward transformation, a two-dimensional array of size 512 * 512 is reduced in multiple steps in order to be stored in another two-dimensional array of size 32 * 32. The input data we used in our experiments are taken from the well-known "lena" picture. In the second phase of the benchmark, the reduction is reversed and afterwards the output is compared to the original data. During the forward transformation the size of the array is reduced by a factor of four in consecutive steps. Each step consists of two loops.

B. Optimization

Even though the input and output buffer and the number of iterations changes between the different steps, the loops between different steps are of the same structure. Therefore all possible optimizations found for the two loops in the first step can be applied for the loops in all the other steps. For the reverse transformation the same applies since it is also based only on two different loop structures. Furthermore, the two loops of each step have a similar structure. One of them is slightly more complex, however. Therefore all optimizations were evaluated for the more complex loop and the results of the optimization were applied to the other loops. The original source code inside the loop is shown in Figure 3. Simulation results of the original and the optimized versions are depicted in Table III.

The first optimization applied on the source code was the reuse of values that were loaded from memory multiple times in the original code. Thus, one load can be saved in each operation. Furthermore, the loads are unconditionalized by executing them before the if-then-else construct. The changed source code is displayed in Figure 4 and the results of a simulation are shown in Table III as version "optimized loads". Before this transformation values were not loaded and stored in a temporary variable, but were loaded every time from the memory. Therefore the DRESC compiler could not schedule those values to be stored in a local RF, but had to schedule a separate load operation for each use. This optimization did not improve the performance regarding the number of cycles, but reduced the number of executed instructions, especially of the load and store operations, and therefore had a direct influence on the energy consumed during the processing of the loop. Since load and store operations can only be scheduled on special functional units on the CGRA with

Version	Cycles	Instructions	IPC	Scheduling Time
original source code	4'547'885	38'656'870	8.5	2.96 s
induction variable strength reduction	4'547'879	36'382'913	8.0	2.65 s
1 time unrolled loop	3'410'927	32'972'096	9.7	11.99 s
2 times unrolled loop	2'273'994	29'561'293	13.0	28.02 s
7 times unrolled loop	2'273'967	25'866'134	11.4	84.19 s
11 times unrolled loop	3'514'319	27'080'725	7.7	226.98 s

TABLE II

Simulation results of different optimizations for the Tiff2BW benchmark on the 4x4 multimedia ADRES instance

```
if (row==0)
  tmp1 = L1[0][(2*row)+1][(2*col)+hor];
else
  tmp1 = L1[0][(2*row)-1][(2*col)+hor];
L2[2*hor][row][col] = L1[0][(2*row)][(2*col)+hor]-
            (L1[0][(2*row)+1][(2*col)+hor]+tmp1)/2;
```

Fig. 3. Original source code of the wavelet transformation

```
L1_0_r_1 = L1_0_r1;
L1_0 = L1[0][(2*row)][(2*col)+hor];
L1_0_r1 = L1[0][(2*row)+1][(2*col)+hor];
if (row==0)
  tmp1 = L1_0_r1;
else
  tmp1 = L1_0_r_1;
L2[2*hor][row][col] = L1_0 - (tmp1 + L1_0_r1)/2;
```

Fig. 4. Source code with optimized loads

Version	Cycles	Instructions	IPC	Scheduling Time
original source code	175'498	2'610'838	14.9	114.21 s
optimized loads	176'520	1'936'019	11.0	221.12 s
copy propagation	143'496	1'747'857	12.2	221.65 s
loop coalesced	82'620	1'156'423	14.0	297.53 s

TABLE III

Simulation results of different optimizations for the wavelet transformation on the 4x4 multimedia ADRES instance

a memory access capability, the reduction of those operations also gave the compiler more freedom to schedule the loop.

Following this source level transformation the loading of the values was rewritten in order to exploit the copy propagation handling of the DRESC. Values needed in the next iteration of a loop are then stored in a local RF and did not need to be loaded again from the memory. The source code change can be seen in the first line of Figure 4 and its improvement for the simulation in Table III as version "copy propagation".

In the final step of the optimization process the outer and inner loops were coalesced in order to increase the processing time spent in the kernel state and to reduce the influence of the loop prologues and the loop epilogues on the total performance of the applications. In software-pipelined loops, the stages of the pipeline have to warm up when the loop is entered (as only stages from already initiated iterations are to be executed), while the pipeline is flushed at the end of the loop when no more new iterations are initiated. These warmup and flushing phases are called prologues and epilogues. The fewer the number of iterations in a loop, the higher the effect of their length on the total processing time of a loop. Thus, it is important to create loops with a high number of iterations to obtain high performance. Loop coalescing, being the combination of nested loops (say with iteration counts i_1 and i_2) into a single loop (then with iteration count $i_1 \times i_2$), is ideally suited for this. The results for the simulation of this final source code can be seen in Table III as version "loop coalesced". After this optimization had been applied, no further improvement was found, so it were these optimizations that were applied to all other loops in the whole wavelet benchmark.

As for the Tiff2BW benchmark the final simulation results for the wavelet transformation for different ADRES architectures are shown in Table III. In the wavelet transformation the number of cycles needed could be reduced to only 50% of the original used cycles and the number of instructions executed was reduced by more than 55%.

V. Architecture Exploration

The final source code versions of the benchmarks were simulated on different architectures of ADRES, as mentioned in Section A. The next section will explain how numbers for the TI DSP c64x were obtained and what optimizations were applied before the benchmarks were compiled for that architecture. In Section B, the obtained numbers will be discussed.

A. Comparison with the TI c64x

In order to evaluate the gathered results the two benchmarks were run on a simulator for a DSP from TI (c64x). For the simulation the Code Composer Studio v3.2 from TI was used and the source code was compiled with the highest optimization parameter for TI's proprietary compiler. All applied optimisations for the ADRES architecture were checked on the DSP and if necessary undone. Furthermore the by the TI compiler generated assembler code was inspected for additional optimisations.

A.1 Tiff2BW Benchmark

Unfortunately the used source code proved to be incompatible with the simulator due to the not supported use of an external library in the Tiff2BW benchmark. In order to enable the simulation on the DSP, the source had to be rewritten. Instead of opening a TIFF image the algorithm will now allocate memory of the same size as the image and fill it with pseudo random numbers. This does not

Benchmark & Architecture	Cycles	Instructions	IPC	Utilisation of the CGRA	Speedup to TI c64x
Tiff2BW					
DSP TI c64x	4'547'947	26'529'081	6		1.00
4x4 multimedia	2'273'967	24'729'217	10.9	68%	2.00
4x4 wireless	3'410'949	26'718'897	7.8	49%	1.33
2x2 alternative	8'811'517	24'444'802	2.8	70%	0.52
4x4_all alternative	2'894'189	32'662'290	11.3	71%	1.57
4x4_4L alternative	2'687'434	32'662'331	12.2	76%	1.69
4x4_16L alternative	2'687'427	21'421'906	11.7	73%	1.69
8x8 alternative	1'421'264	35'814'939	25.2	39%	3.20
Wavelet					
DSP TI c64x	8'765'163	26'973'231	3.08		1.00
4x4 multimedia	2'095'528	28'346'939	13.5	84%	4.18
4x4 wireless	2'623'134	30'137'922	11.5	72%	3.34
2x2 alternative	6'767'899	26'117'944	3.9	98%	1.30
4x4_all alternative	3'091'148	35'451'612	11.5	72%	2.84
4x4_4L alternative	2'885'585	35'022'239	12.1	76%	3.04
4x4_16L alternative	2'792'529	34'202'225	12.2	76%	3.14
8x8 alternative	1'627'106	46'342'177	28.5	45%	5.39

TABLE IV
COMPARISON OF THE IMAGE PROCESSING BENCHMARKS FOR DIFFERENT ADRES ARCHITECTURES AND THE TI C64X

change the produced simulation results since the benchmark will still process the same amount of elements and will still perform the same amount of operations, all in the same order.

A.2 Wavelet Transformation

For the wavelet transformation the initial run of the source code optimized for ADRES on the simulator from TI showed that the DSP stayed far below the expected performance. All applied optimizations were checked for performance on the DSP. First, all the variables and data had to be restricted to 16-bit values. Also, because the loop coalescing introduces division operations, which are not supported by a single instruction in the TI instruction set, the coalescing of the loops had to be undone because it otherwise resulted in slow instruction sequences implementing division.

B. Simulation Results

After the changes on the source code described in the previous sections both applications were simulated on different ADRES architectures as well as on the DSP c64x with an optimized source code version for each architecture including the TI c64x. The results of those simulations are shown in Table IV. Due to the higher data dependency in the wavelet transformation the DSP performs worse for this benchmark than in the Tiff2BW benchmark. The ADRES architecture can profit in this specific case more from its copy propagation.

In all cases except for one the ADRES architecture and its corresponding compiler DRESC performs better than the TI c64x. Only the 2x2 architecture is worse than the DSP for the Tiff2BW benchmark. This specific ADRES architecture has only half of the functional units of the TI c64x and the parallelism of the benchmark is fairly easy to exploit on the DSP c64x. For the wavelet transformation even the trade off of less FUs is compensated in a 2x2 architecture and the 2x2 ADRES architecture outperforms the TI c64x. Especially in the wavelet transformation a significant gain can be reached. For the "4x4 multimedia" architecture the number of cycles was reduced to less than 25% compared to the TI c64x.

Between the different ADRES architectures the "4x4 multimedia" instance provides the best results. The "4x4 wireless" instance performs worse even with the larger data width, because this data width is not needed in the mapped benchmarks and SIMD for those benchmarks can not be explored with this instance. Furthermore the shared read-only data memories (that can only store a limited number of immediate operands to instructions) limit the performance for the two benchmarks. This is mainly caused by the compiler and will be addressed in the future compiler releases. The alternative 4x4 architectures[3] show that for those specific benchmarks the tradeoffs for saving resources (local RFs, connections between the resources) is minimal. As already mentioned before the 8x8 architecture stays below the expected performance. This may be caused by a lack of scalability of the simulated-annealing scheduling algorithm, or by the relatively lower relative connectivity of such a large architecture, or by yet another reason. Future research in improveming the DRESC compiler for better support of large ADRES architectures will give more insights in this behavior.

C. Power Measurements

In order for the power dissipation estimation tool flow to be executed, the VHDL code automatically generated for the target architecture of the ADRES template, the "4x4 multimedia" instance was to be synthesized, placed, routed, and simulated. We obtaining switching activity of the nets by a cycle true simulation. The switching activity is afterwards annotated on the synthesized design, thus leading to an accurate power estimation.

The experiments for the power measurements were done for the Tiff2BW benchmark as well as for the Wavelet transformation. To be simulated the total amount of memory accessed needed to be reduced. In order to reduce the total amount of memory used in both benchmarks, the size of the processed image was reduced. This reduction had no influence on the power measurements, as the activity inside the main loops is not altered. Only the length of the loops was altered.

ADRES component	Tiff2BW in mW	Wavelet in mW
Icache	9.233	8.492
Data MEM	17.330	11.180
CORE	37.270	57.140
CGRA mode control	8.916	14.025
CGRA FUs	9.513	15.175
VLIW mode control	0.205	0.198
VLIW FUs	0.659	0.780
Clock Tree	12.700	13.640
Total Consumption (processor level)	71.690	84.590
DSP c64x (given by TI [9]) @720MHz; 1.2V	1700.000	1700.000

TABLE V
POWER NUMBERS FOR THE "4X4 WIRELESS" ADRES INSTANCE FOR BOTH BENCHMARKS

Table V shows the gathered power numbers for the 4x4 multimedia ADRES instance. The Icache is only used in VLIW mode of the ADRES to fetch the VLIW instructions and due to the fact, that the transformations will be done in CGRA mode, this power number only accounts for the start of the application and small parts of the application to prepare for the transformation. The power number for the Data MEM represents the power consumed for the load and store operations in the application. The core power numbers correspond to the power consumed by the ADRES processing unit. Since there is only a small amount of processing time spent in VLIW mode the power number for those units in the ADRES processing unit are relatively small. The main power consumption for both benchmarks is caused by the CGRA control unit and by the CGRA functional units. In general 50% of the power consumption is caused by the memories such as Icache, Data MEM and CGRA control unit, which is the instruction memory for the CGRA mode of the ADRES.

Compared to the power numbers of the TI DSP c64x, taken from [9] and also shown in Table V, the given ADRES architecture performs below 5% of the DSP's power level. The 4x4 multimedia instance will operate on 45.6 MOPS per mW for the Tiff2BW benchmark and on 47.9 MOPS per mW for the wavelet transformation.

VI. Conclusion

Our results show that the ADRES architecture and the DRESC compiler are highly suitable for image processing. Due to its data independency in the Tiff2BW benchmark and the limited data dependency between iterations in the wavelet transformation the DRESC compiler can nearly fully exploit the capacity of the CGRA. Limitations made to the benchmarks do not affect the initial performance of the benchmarked transformation itself. Most of the limitations are due to external causes such as problems with the IMPACT tool or problems caused by the TI compiler. A few problems in the wavelet transformation were caused by the DRESC compiler itself, but they have only a minor influence in the performance and will be solved in the near future.

It has been shown that an improvement of factor two can be reached on a 4x4 ADRES architecture compared to the DSP c64x from TI for the Tiff2BW benchmark and an improvement of factor 4 for the wavelet transformation regarding the performance. As for the power numbers experiments have proven, that the 4x4 multimedia ADRES instance will work below 5% of the power level of the TI c64x. An 8x8 ADRES template will increase the performance even more, but it does not reach the expected performance and the capacity of the CGRA is not fully exploited. Further research will be done on a better scheduling process for large CGRA structures.

References

[1] Sami Khawam Mark Milward Ying Yi Adam Major, Ioannis Nousias and Tughrul Arslan. H.264/avc in-loop de-blocking filter targeting a dynamically reconfigurable instruction cell based architecture. 17th International Conference on Field Programmable Logic and Applications.

[2] B. Bingfeng Mei; Veredas, F.-J.; Masschelein. Mapping an h.264/avc decoder onto the adres reconfigurable architecture. pages 622 – 625, Aug 2005. Field Programmable Logic and Applications, 2005. International Conference.

[3] Frank Bouwens. Power and performance optimization for adres. Master's thesis, Delft University of Technology, The Netherlands, August 2006.

[4] R. Hartenstein. A decade of reconfigurable computing: a visionary retrospective. 2001.

[5] The IMPACT Group. http://www.crhc.uiuc.edu/Impact/.

[6] MiBench Benchmark Suite http://www.eecs.umich.edu/mibench/.

[7] Texas Instruments Inc. http://www.ti.com/.

[8] Sundeep Singh Frank May Mahendra Kumar, Angamuthu Ganesan and Jrgen Becker. H.264 decoder at hd resolution on a coarse grain dynamically reconfigurable architecture. 17th International Conference on Field Programmable Logic and Applications.

[9] Gustavo Martinez. Tms320c6455/c6454 power consumption summary. Dec 2006. http://focus.ti.com/lit/an/spraae8a/spraae8a.pdf.

[10] Bingfeng Mei, Serge Vernalde, Diederik Verkest, Hugo De Man, and Rudy Lauwereins. ADRES: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix. In *Proc. of Field-Programmable Logic and Applications*, pages 61–70, 2003.

[11] Bingfeng Mei, Serge Vernalde, Diederik Verkest, Hugo De Man, and Rudy Lauwereins. Exploiting loop-level parallelism for coarse-grained reconfigurable architecture using modulo scheduling. *IEE Proceedings: Computer and Digital Techniques*, 150(5), september 2003.

[12] W.; Derudder V.; Bougard B. Novo, D.; Moffat. Mapping a multiple antenna sdm-ofdm receiver on the adres coarse-grained reconfigurable processor. pages 473 – 478, Nov 2005. Signal Processing Systems Design and Implementation, 2005. IEEE Workshop.

[13] Hartej Singh, Ming-Hau Lee, Guangming Lu, Fadi J. Kurdahi, and Nader Bagherzadeh. Morphosys: an integrated reconfigurable system for data-parallel and computation-intensive applications. pages 465 – 481, May 2000. IEEE Transactions on Computers.

[14] A. Skodras, C. Christopoulos, and T. Ebrahimi. The JPEG2000 still image compression standard. *IEEE Signal Processing Magazine*, 9(7):36–58, September 2001.