Speedups and Energy Savings of Microprocessor Platforms with a Coarse-Grained Reconfigurable Data-Path

Michalis D. Galanis, Gregory Dimitroulakos, and Costas E. Goutis
VLSI Design Laboratory, Electrical & Computer Eng. Dept., University of Patras, Greece
e-mail: {mgalanis, dhmhgre, goutis}@ece.upatras.gr

Abstract

This paper presents the performance improvements and the energy reductions by coupling a high-performance coarse-grained reconfigurable data-path with a microprocessor in a generic platform. The data-path has been previously introduced by the authors. It is composed by computational units able to realize complex operations which aid in improving the performance of time critical application parts, called kernels. A design flow is proposed for mapping high-level software descriptions to the microprocessor system. Eight real-life applications are mapped on three different instances of the system. Significant overall application speedups, relative to a software-only solution, ranging from 1.74 to 3.94 are reported being close to theoretical speedup bounds. Average energy savings of 59% are achieved, while the reduction in the system energy-delay product ranges from 66% to 92%.

1. Introduction

Embedded systems have been proposed that extend the computational capabilities of a microprocessor core by coupling it with reconfigurable logic [1]-[8]. Reconfigurable hardware is used to implement computational intensive parts of the applications, called *kernels*. The microprocessor core executes non-critical sequential code parts and provides software programmability. The spatial parallelism present in kernels is exploited by the abundant Processing Elements (PEs) of the reconfigurable hardware, resulting in performance improvements. There is a growing interest in reconfigurable systems from designers, especially with the introduction of commercial devices that combine reconfigurable logic with one or more instruction-set processors [3], [4].

Reconfigurable hardware has been widely associated with Field Programmable Gate Arrays (FPGAs). An FPGA consists of an array of programmable logic cells, executing bit-level operations, with a grid of interconnect lines running among them. Both the function and interconnection among the logic cells can be programmed after fabrication. FPGAs are more effective in realizing bit-level operations. However FPGAs are not the only type of reconfigurable logic. Several coarse-grained reconfigurable architectures have been introduced and successfully built [1], [5], [6], [7], [8]. They consist of a large number of PEs with word-level data bit-widths (like 16-bit ALUs) connected with a reconfigurable interconnect network. The coarse-grained PEs exploit better the word-level parallelism of many DSP applications than the FPGAs do. Their coarse granularity greatly reduces the delay, area, power consumption and reconfiguration time relative to FPGA logic at the expense of flexibility [1].

In this work, we propose the coupling of microprocessor cores with a high-performance coarse-grained reconfigurable data-path previously presented in [9]. The computational resources of the Reconfigurable Data Path (*RDP*) are able to implement complex arithmetic structures. Research in High-Level Synthesis (HLS) [10], [11] and in Application Specific Instruction Processors (ASIPs) [12], [13] have proven that the use of complex computational structures, called *templates* or *clusters*, instead of using only primitive ones (like a single ALU) in data-paths improves performance. A template may be a specialized hardware unit or a group of chained units. Chaining is the removal of the intermediate registers between the primitive units improving the total delay of the combined units. The ability of the considered RDP to realize complex operations aids in improving performance as it was shown in [9]. A design flow is introduced for efficiently mapping applications described in C language to the reconfigurable system. The flow consists of the following steps: (a) a profiling procedure, (b) Intermediate Representation (IR) creation of the kernel code, (c) optimizations to the IR, (d) mapping of the kernels to the RDP, and (e) compilation of the non-critical segments to the microprocessor. Parts of the design flow have been implemented as prototype tools.

Design flows for systems integrating coarse-grained reconfigurable logic have been presented [1]. However, few of those works consider the mapping of realistic applications and examine the performance and energy consumption improvements. Research activities that consider the mapping of real-life applications on coarse-grained reconfigurable based systems are overviewed. In [8], a PipeRench architecture clocked at 100MHz and composed by 8-bit PEs and 53 128-bit stripes, improved on average the execution time by 12% and 7.2% for the Pretty Good Privacy (PGP) data encryption algorithm and

This work was partially funded by the Alexander S. Onassis Public Benefit Foundation

for JPEG, respectively, relative to the execution on an UltraSparcII running at 300 MHz. The compilation framework of [14] achieved the acceleration of a wavelet compression and Prewitt detection on the MorphoSys architecture over the execution on a Pentium III machine. In [15], it is shown that a hybrid architecture composed by an ARM926EJ-S and an 8x8 Reconfigurable Array similar to MorphoSys [5], executes 2.2 times faster an H.263 encoder than a single ARM926EJ-S processor. The mapping flow for the reconfigurable ADRES architecture was applied to an MPEG-2 decoder in [16]. The kernel and the overall application speedup over an 8-issue VLIW processor were 4.84 and 3.05, respectively. In [7], an H.264/AVC decoder was mapped on an 8x8 array achieving application speedup of 1.88. To our knowledge, the only work that studies the effect in the energy by extending a RISC core with a coarse-grained reconfigurable unit is the one in [17]. The experiments showed that the performance is improved 2.5 times on average while the energy-delay product is 60% on average smaller than the one of the baseline RISC. The power consumption of the reconfigurable logic is estimated using the Wattch power estimation framework that mainly targets superscalar processors.

This work presents the performance improvements as well as the energy reductions by executing time critical kernels of realistic benchmarks on the coarse-grained RDP. Eight real-world DSP applications are mapped on three systems employing 32-bit ARM processors. The applications' execution times are estimated using the proposed design flow. Typical power values are considered for the ARM processors. The RDP is described in synthesizable VHDL and its power consumption is estimated using commercial tools. Important application speedups are reported as the design flow accelerates each application close to ideal speedups. The performance improvements lead in significant energy savings, while the energy-delay product values are substantially reduced over an all-software execution.

The rest of the paper is organized as follows: section 2 presents the architecture of the microprocessor platform and the proposed RDP architecture. Section 3 describes the design flow and outlines the mapping procedure for the reconfigurable data-path. Experimental results are given in section 4, while section 5 concludes this paper.

2. System architecture

2.1. Platform description

Figure 1 shows an overview of the microprocessor platform considered in this work. The System-on-Chip (SoC) architecture includes an embedded microprocessor, Reconfigurable Logic (*RL*) and system RAM. The RL is composed by a Coarse-Grained Reconfigurable Data Path (*RDP*) introduced in [9] and a set of memory-mapped coprocessor data registers for exchanging data with the microprocessor. The system RAM stores data, instructions for the microprocessor execution and configurations for the RL. The RDP acts as a coprocessor to the microprocessor and accelerates time critical software parts of an application by exploiting the Instruction Level Parallelism (ILP) of these parts. The microprocessor, typically a RISC like an ARM, executes control-dominant software parts.

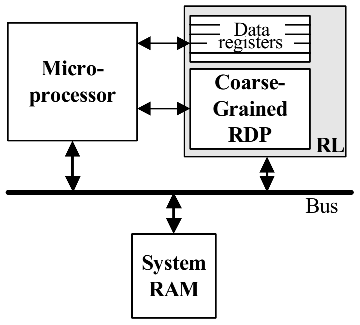

Figure 1. Diagram of the microprocessor SoC platform.

Data communication between the Coarse-Grained Reconfigurable Data Path and the CPU uses shared-memory mechanism. The shared memory is comprised of the shared system RAM and the memory-mapped coprocessor registers within the reconfigurable logic. Scalar variables, either live-in or live-out ones, are identified utilizing data-flow analysis and they are exchanged via the shared registers. Global variables and data arrays are allocated in the system RAM. Both the microprocessor and the RDP have access to the shared memory. The communication process used by the processor and the RDP preserves data coherency by requiring the execution of the processor and the RDP to be mutually exclusive. The mutual exclusive execution simplifies the programming since complicated analysis and synchronization procedures are not required. The system architecture is also simpler with the mutual exclusive execution because the processor and the RDP will never access simultaneously the same data memory.

A software kernel is replaced with code that enables the reconfigurable logic. When a kernel is reached in software, the RDP is activated and the proper configuration is loaded for executing the critical code. Furthermore, the live-in local variables are transferred to the shared data registers in the reconfigurable logic. Then, the microprocessor enters a low-power state. After the completion of the kernel execution, the RDP informs the processor waking-up from its low-power state, writes the live-out variables in the shared registers, and writes global variables and array data located in the shared RAM. Then, the execution of the software is continued on the CPU and the RDP enters a low-power idle state for reducing system energy. It is mentioned that waking-up a microprocessor for its power-down sleep mode requires from few cycles to a few dozen, depending on the microprocessor. In either situation, these cycles are only consumed after a kernel on the RDP completes (and not at every iteration of the kernel) and so it is typically insignificant compared to the thousand of cycles required for the kernel execution.

2.2. RDP architecture

An outline of the data-path is shown in Figure 2. The considered high-performance coarse-grained reconfigurable data-path and the respective mapping methodology have been introduced in [9]. The proposed coarse-grained RDP consists of a set of word-level hardwired Coarse-Grained Components (CGC) units, a reconfigurable interconnection network, a register bank, configuration RAMs (CR), Address Generation Units (AGUs) and a control unit. The data-width of the RDP is typically 16-bits, although different word-level bit-widths are supported.

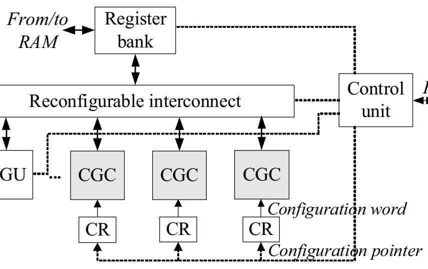

Figure 2. Outline of a CGC-based data-path.

The control unit manages the execution of the RDP every cycle by configuring the interconnect, the AGUs, the register bank and the CGCs. The microprocessor sets the control unit for proper execution of a kernel on the data-path. The interconnection network provides full connectivity among the CGCs and connectivity to the register bank. For a small number of CGC units (up to four units) present in the data-path, a full crossbar is used. When a large number of CGCs is employed, a hierarchical network is utilized. The register bank stores intermediate values among computations and data fetched from the system RAM of the SoC. The AGUs enable the data accesses to the system RAM.

The configuration (context) RAMs stores few context words locally. During the execution of a kernel, the configuration RAMs of the CGCs are indexed by a central Configuration Pointer, set by the control unit of the RDP, and a proper configuration word is loaded allowing dynamic reconfiguration of each CGC within a cycle. The configuration word defines the functionality of the CGC. Each CGC can be configured to execute a different complex operation in a cycle according to the contexts stored in the configuration RAM. The contexts can also be loaded from the system RAM at the cost of extra delay, if the configuration RAMs are not large enough to store the configurations of a kernel. The control unit manages the loading of the contexts from the system RAM. The system RAM stores the whole configuration for setting up the RDP for the execution of the kernels. At the initialization phase of the RDP, the contexts are loaded from the system RAM into the configuration RAMs similar as in FPGAs.

A CGC unit is an *n*x*m* array of Processing Elements (PEs), where *n* is the number of rows and *m* the number of columns. In Figure 3a, such a CGC (called hereafter as 2x2 CGC) with 2 PEs per row and 2 PEs per column is illustrated. The PEs in the first row are connected to the second row PEs with multiplexer-based flexible connections. Each CGC PE contains a word-level ALU-multiplier unit. The ALU part performs shifting, arithmetic (add/subtract), and logical operations. At each cycle, the ALU-multiplier unit is properly configured to perform either a multiplication or an ALU operation. The four inputs (*in1*, *in2*, *in3*, *in4*) are connected to the register bank, the four additional inputs (*A*, *B*, *C*, *D*) are connected to the register bank or to another CGC, the two outputs (*out1*, *out2*) are also connected to the register bank or to another CGC, and the two outputs (*out3*, *out4*) store their values in the register bank. An *n*x*m* CGC has an analogous structure. A local configuration register stores the configuration word broadcasted from the corresponding Configuration RAM of the CGC. It is mentioned that for a 2x2 CGC, a context word of 36-bit is adequate for configuring it, whereas for a 3x2 CGC the word has size of 62 bit.

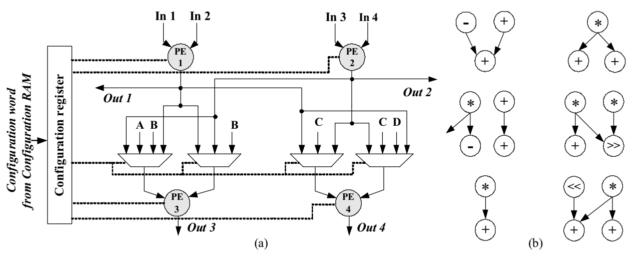

Figure 3. (a) Architecture of the 2x2 CGC, (b) Examples of complex operations realized by the 2x2 CGC.

The flexible connections among the PEs inside a CGC allows in easily realizing any desired operation combination, as the ones proposed in [10]-[13], by properly configuring the multiplexers of the CGC. Examples of complex operations (templates) realized by the 2x2 CGC are shown in Figure 3b. Thus, since a CGC can implement templates by properly configuring the connections inside the CGC, high-performance is achieved. In [9], it was shown that an average performance improvement of 44%, relative to existing high-performance data-path, was accomplished with CGC-based data-paths. This improvement is due to the exploitation of chaining of operations inside the CGCs (intra-CGC chaining) and inter-CGC chaining due to the direct connections among the CGCs.

3. Design flow

The flow for mapping applications, described in C, on the processor/RDP architecture is illustrated in Figure 4. Initially, a profiling procedure outputs the kernels and the non-critical parts of the source code. For performing profiling, standard debugger/simulator tools of the development environment of a specific processor can be utilized. For example, for the ARM processors, the instruction-set simulator (ISS) of the ARM RealView Developer Suite (RVDS) can be typically used. Kernels are considered those code segments that contribute more than a certain amount to the total application's execution time on the processor. For instance, parts of the code that account 10% or more for the application's time can be characterized as kernels.

The kernels are moved for execution on the RDP. The Intermediate Representation (IR) of the kernel source code is created using a compiler front-end. A representation widely used in reconfigurable systems is the Control Data Flow Graph (CDFG). In this work, we utilize a hierarchical CDFG for modeling data and control-flow dependencies. The control-flow structures, like branches and loops, are modeled through the hierarchy, while the data dependencies are modeled by Data Flow Graphs (DFGs). For generating the CDFG IR from C source code, we have utilized the SUIF2/MachineSUIF compiler infrastructures. Existing and custom-made compiler passes are used for the CDFG creation.

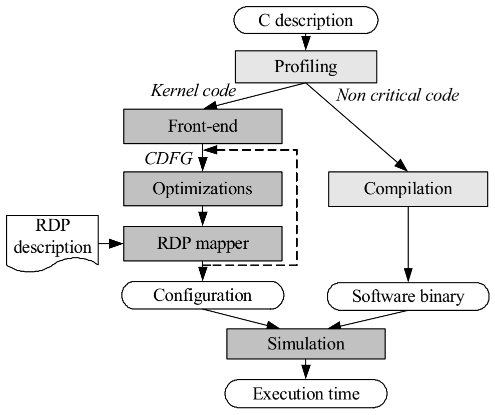

Figure 4. Design flow for the processor/RDP system.

Optimizations are applied to the kernels for efficient mapping on the RDP. Examples of optimizations used in the flow are dead code elimination, common sub-expression elimination and constant propagation. Additionally, operations inside the kernels that cannot be directly executed on the PEs of the CGC units are transformed into series of supported operations. The divisions are transformed to shifts, while a square root computation can be performed by the CGCs using a method, like the Friden algorithm [18] that has been implemented in the proposed flow. MachineSUIF compiler passes have been developed for the automatic application of the optimizations on a kernel's CDFG.

The optimized kernels are mapped on the RDP, for improving performance, utilizing the algorithm outlined in section 3.1 which is the core of the design flow. A prototype tool in C++ has been developed for implementing the mapping procedure. The second input to the mapping process is a description of the RDP. The mapping tool outputs the execution cycles and the configuration of the CGC-based data-path. A feedback script is included in the flow for optimizing the performance of the kernels executed on the RDP.

The non-critical source code is compiled using a compiler for the specific processor. The system's performance is estimated via simulation having as inputs the software binary of the processor and the configuration of the RDP. The dark grey boxes in Figure 4 represent the automatic procedures modified or created by the authors for the specific flow, while the light grey boxes the external tools used.

The total execution time required for executing an application on the generic system is:

$$Time_{system} = Time_{proc} + Time_{RDP} \qquad (1)$$

where $Time_{proc}$ represents the time for executing non-critical software parts on the processor, and $Time_{RDP}$ corresponds to the execution time of the kernels on the RDP. The communication time between the processor and the RDP is included in the $Time_{proc}$ and in the $Time_{RDP}$.

3.1. RDP mapper

The flow of the mapping method for the CCG data-path is shown in Figure 5. The input is the CDFG of a kernel decided by the detection procedure to be executed on the coarse-grained reconfigurable hardware. The proposed mapping procedure traverses the kernel's CDFG and maps one DFG at a time. A DFG is scheduled using the developed scheduler for the CGC data-path. In our case, scheduling is a resource-constrained problem with the goal of execution cycle count minimization, since the number and type of CGCs (e.g. three 2x2 CGCs) in the data-path is constant and input to the mapping procedure. A proper list-based scheduler has been developed. The priority function of the scheduler is derived by properly labeling the DFG operations. Particularly, the operations are labeled with weights of their longest path to the sink operation of the DFG, and they are ordered by decreasing weight. The most urgent operations are scheduled first. The arithmetic resource constraints for the scheduler are determined by the total number of PEs at the first rows of all the CGCs in the data-path. If there are *p n*x*m* CGCs in the data-path, there are $p \cdot m$ PEs in the first rows, since each CGC row consists of m PEs. Thus, $p \cdot m$ primitive operations (ALU and/or multiplications) can be executed in parallel at each clock cycle of the schedule. For example, if there are three 2x2 CGCs in the data-path, six operations can be executed in parallel at every cycle of the schedule.

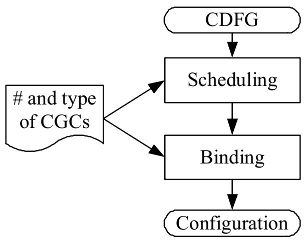

Figure 5. Mapping procedure for the RDP.

The input to the binding step is the scheduled DFG. The pseudo-code of the binding algorithm is shown in Figure 6. After binding, the overall execution delay of the DFG is measured in clock cycles having period T_{CGC}. This period is set for having unit execution delay for the CGCs. The CGC binding algorithm maps row-wise the DFG operations to the CGC PEs. A term called *CGC_index* is defined that it is related to the clock period T_{CGC} after the binding is performed. This term represents the current row of CGC's PEs that bind the DFG operations. The *CGC_index* takes the values from 0 and n-1, since a CGC consists of n rows of PEs. The algorithm covers the scheduled operations for *CGC_index* equal to 0 or until there are no DFG operations left uncovered. Then, it proceeds to the next value of *CGC_index* till equal to n-1, if there are any uncovered operations left. This procedure is repeated for every CGC in the data-path. A CGC is not utilized, when they are no uncovered operations left. Also, a CGC is partially utilized when there is no sufficient number of operations left and the mapping to CGC process (*map_to_CGC*) has already been started for this CGC. The mapper outputs the execution cycles of the kernel and the configuration of the CGC-based RDP. For more detailed description about the mapping method, the reader is referred to [9].

```
while (the DFG is not covered)
  for the number of CGCs
    for (CGC_index=0; CGC_index <n; CGC_index ++)
      while (col_idx <  number of ops in a row &&
                 col_idx < number of DFG nodes not covered)
        map_to_CGC(DFG_node, CGC_index, col_idx)
      end while;
    end for;
  end for;
end while;
```

Figure 6. CGC binding pseudocode.

4. Results

This section presents the performance improvements and the energy savings by employing the CGC-based data-path on microprocessor systems.

4.1. Platform's processing units

The CGC-based RDP has a data-width of 16-bit and it is composed by three 2x2 CGCs. In our previous work [9] it was inferred that a small number of CGCs with a value of $2 \leq n \leq 3$ and $2 \leq m \leq 3$ are adequate for improving performance. The local configuration RAMs are implemented as SRAMs and the size of each of them is 32 contexts of 36-bit. The register bank has a size of 32 words of 16-bit. The RDP was described in register-transfer level (RTL) VHDL. For the SRAMs, optimized components from TSMC 0.13μm SRAM generators of Artisan Components [22] were used. The Synopsys synthesis and power estimation tools were utilized to obtain delay, area and power estimates for a 130nm standard cell TSMC CMOS library. It was found that the critical path delay allows the maximum clock frequency to be 150MHz. However, we set the frequency of the RDP to be 100MHz for reducing energy consumption. The average power consumption of the RDP was estimated using simulation vectors from kernels of the applications used in the experiments. It was reported that the RDP consumes 0.72mW/MHz at 1.2V. Thus, the power consumption of the RDP is 72.0 mW at 100 MHz.

The embedded CPU used in the platform is an ARM RISC core. Three different architectures of 32-bit ARM cores are coupled each time with the 4x4 CRA. These processors are: (a) an ARM7 clocked at 100 MHz, (b) an ARM9 clocked at 200 MHz, and (c) an ARM10 having clock frequency of 300 MHz. These frequencies can be supported by the considered ARM processors. So, the CPUs are clocked at integer multiples of the operating frequency of the CRA for simpler synchronization between the CPU and the CRA. The bus in each system is an AMBA AHB bus [19].

The eight applications were compiled to generate binary files for the ARM processors using the highest level of software optimizations. The ARM RealView Developer Suite (RVDS) (version 2.2) was used for calculating the execution time of application parts for each one of the three processors. Typical average mW/MHz values are considered for the ARM processors in 130 nm at 1.2V [19]. These are: 0.20 mW/MHz for the ARM7, 0.45 mW/MHz for the ARM9 and 0.60 mW/MHz. Thus, the power consumption for the ARM7 is 20.0mW at 100MHz, for the ARM9 is 90.0mW at 200MHz, while for the ARM10 is 180.0mW at 300MHz.

4.2. Benchmarks and profiling results

The eight real-world DSP applications, described in C language, used in the experiments are given in Table 1. A brief description of each application is given in the second column, while in the third one the input sequence used is presented.

Table 1. Applications' characteristics

Application	Description	Input
JPEG enc.	Still-image JPEG encoder	256x256 byte image
OFDM trans.	IEEE 802.11a OFDM transmitter	4 payload symbols
Compressor	Wavelet-based image compressor	512x512 byte image
Cavity det.	Medical imaging technique	640x400 byte image
Edge det.	Edge detection in images	128x128 byte image
JPEG dec.	Still-image JPEG decoder	227x149 byte image
GSM enc.	Speech encoder	clinton.pcm
GSM dec.	Speech decoder	clinton.pcm.run.gsm

The results from profiling the eight applications on the ARM7 processor, using the ARM RVDS tool, are presented in Table 2. The threshold for detecting kernels was set to the 10% of the total application's execution time. It was observed that a threshold smaller than 10% leads to trivial additional improvements when the identified kernels are mapped on the CGC data-path. The *Total size* corresponds to the application's static size in terms of instructions bytes, while the *% size* to the percentage of the kernels' contribution to the total static size. The *% time* is the percentage of the execution time spent in the kernels. The *Ideal speedup* is the theoretical maximum speedup, according to Amdahl's law, if the application's kernels were ideally executed on the CGC-based RDP in zero time. The ideal speedup equals 100 / (100 - *%time*). The number of the kernels detected in each application is also given. The kernels of the eight applications are innermost loops and they consist of word-level operations (ALU, multiplications, shifts) that match the granularity (data bit-width) of the PEs in the CGCs. We mention that the detected loops are also kernels for the ARM9 and ARM10 processors.

Table 2. Results from profiling the benchmarks on ARM7

Application	Total size	% size	% time	Ideal speedup	# of kernels
JPEG enc.	36,592	10.4	74.7	3.96	4
OFDM trans.	15,579	9.0	71.8	3.54	4
Compressor	12,835	4.8	60.2	2.51	4
Cavity det.	12,039	7.4	58.0	2.38	4
Edge det.	9,771	10.6	61.9	2.61	2
JPEG dec.	56,987	2.3	76.0	4.17	4
GSM enc.	125,383	1.1	67.2	3.05	2
GSM dec.	113,383	0.9	64.5	2.82	1
Average		**5.8**	**66.8**	**3.13**	
Geo. mean		***4.1***	***66.5***	***3.07***	

From these results, it is inferred that an average of 5.8% of the code size, representing the kernels' static size, contributes 66.8% on average to the total execution time. The geometrical means of the *% size* and *% time* are also given. Furthermore, the average ideal speedup for the ARM7 systems equals 3.13. Thus, it is deduced that important overall application speedups will come from accelerating few small kernels.

4.3. Speedups

The execution times and the overall application speedups for the eight applications are presented in Table 3. $Time_{sw}$ represents the software execution time of the whole application on a specific microprocessor (*Proc.*). The ideal speedup (*Ideal sp.*) is the application speedup that would ideally be achieved, according to Amdahl's Law, if application's kernels were executed on the RDP in zero time. $Time_{system}$ corresponds to the execution time of the application when executing the critical code on the CGC data-path. All execution times are normalized to the software execution times on the ARM7. The *Sp.* is the estimated application speedup, after utilizing the developed design flow, over the execution of the application on the microprocessor. The estimated speedup is calculated as:

$$Sp = Time_{sw} / Time_{system} \quad (2)$$

The average values, as well as, the geometrical means of the speedups are also illustrated.

Table 3. Execution times and application speedups

Application	Proc.	$Time_{sw}$	Ideal sp.	Processor/RDP	
				$Time_{system}$	Sp.
JPEG enc.	ARM7	1.000	3.96	0.272	3.68
	ARM9	0.461	3.24	0.148	2.93
	ARM10	0.301	3.16	0.092	2.67
OFDM trans.	ARM7	1.000	3.54	0.305	3.28
	ARM9	0.485	3.43	0.146	3.13
	ARM10	0.344	3.23	0.089	3.13
Compressor	ARM7	1.000	2.51	0.450	2.22
	ARM9	0.424	2.32	0.197	2.02
	ARM10	0.283	2.21	0.132	1.75
Cavity det.	ARM7	1.000	2.38	0.493	2.03
	ARM9	0.480	2.29	0.241	1.87
	ARM10	0.355	2.17	0.166	1.74
Edge det.	ARM7	1.000	2.61	0.407	2.46
	ARM9	0.498	2.54	0.198	2.36
	ARM10	0.367	2.49	0.132	2.27
JPEG dec.	ARM7	1.000	4.17	0.254	3.94
	ARM9	0.418	3.85	0.108	3.64
	ARM10	0.273	3.64	0.067	3.32
Gsm enc.	ARM7	1.000	3.05	0.348	2.87
	ARM9	0.426	2.93	0.145	2.77
	ARM10	0.295	2.88	0.090	2.67
Gsm dec.	ARM7	1.000	2.82	0.364	2.75
	ARM9	0.422	2.77	0.146	2.71
	ARM10	0.292	2.74	0.089	2.67
Average			**2.96**		**2.70**
Geo. mean			***2.90***		***2.64***

From the results given in Table 3, it is evident that significant overall performance improvements are achieved when critical software parts are mapped on the CGCs. These speedups range from 1.74 to 3.94. It is noticed from Table 3 that the largest overall application performance gains are achieved for the ARM7-based architectures as the ARM7 has the lowest clock frequency and exhibits the highest Cycles Per Instruction (CPI) among the three ARM-based systems. The average application speedup of the eight DSP benchmarks for the ARM7 systems is 2.90, for the ARM9 is 2.68, while for the ARM10 systems is 2.53. Thus, even when the CGC-based data-path is coupled with a modern embedded processor, as the ARM10, which is clocked at a higher clock frequency, the application speedup over the execution on the processor core is significant.

The average overall application speedup for all the microprocessor systems is 2.70. Such amounts of application speedups were also considered as important in previous works considering a processor coupled with a coarse-grained reconfigurable logic as in [7], [15], [16]. From Table 3, it is also inferred that the reported speedups for each application and for each processor type are close to theoretical speedup bounds, especially for the case of the ARM7 systems. Specifically, the average estimated speedup for all the systems is 8.8% smaller than the average ideal speedup. Thus, the proposed design flow quite effectively utilized the processing capabilities of the CGC-based data-path for improving the overall performance of the applications near to the ideal speedups.

While experimenting with the benchmarks of this paper, we found that that few parts of each application can be executed in parallel on the processor and on the RDP. A marginal performance increase relative to the mutual exclusive execution was also reported. Such minor improvements cannot offset the benefits of the simpler programming of the system architecture due to the exclusive execution.

4.4. Energy savings

The energy savings by employing a coarse-grained RDP in a microprocessor system are presented next. For estimating the total energy of the system the following formula is used:

$$E_{total} = E_{proc} + E_{RDP} + E_{mem\_icon}$$

$$E_{proc} = Time_{proc} \times P_{active\_proc} + Time_{RDP} \times P_{idle\_proc}$$

$$P_{idle\_proc} = 0.25 \times P_{active\_proc}$$

$$E_{RDP} = Time_{RDP} \times P_{active\_RDP} + Time_{proc} \times P_{idle\_RDP}$$

$$P_{idle\_RDP} = 0.10 \times P_{active\_RDP}$$

$$E_{mem\_icon} = Time_{system} \times P_{mem\_icon}$$

$$Time_{system} = Time_{proc} + Time_{RDP}$$

$Time_{proc}$ is the time of non-critical software parts on the microprocessor, $Time_{RDP}$ is the execution time of the kernels on the RDP, P_{proc} is the power of the microprocessor, P_{RDP} is the power of the RDP and P_{mem_icon} is the power consumption of the system RAM and of the system interconnection buses. In the above formulas, it is considered that the low-power idle mode (standby) of the RDP consumes 10% of the power of its active state as in standby mode the power is primarily due to current leakage. In contemporary CMOS processes, leakage can account for 10% to 30% of the active power [20]. The active power is when the logic (either microprocessor or reconfigurable hardware) evaluates. The power-down mode of the microprocessor dissipates 25% of its power when it is active as in [21]. It is assumed that the systems have P_{mem_icon} of 180 mW. This value was estimated from commercial [19] and academic microprocessor platforms implemented at 130nm as well as from commercial SRAM modules [22] since the system RAM is considered to be a multi-banked SRAM. The values of P_{proc} and of P_{RDP} are given in section 4.1. The formula for the total energy represents a weighted average of the system energy during software execution and the system energy during hardware execution.

Figure 7 illustrates the normalized energy consumptions for the three ARM-based platforms. The energy values are normalized to the software-only execution of each application on the microprocessor. From the presented results it is inferred that significant energy savings are achieved by executing critical kernels on the proposed CGC-based RDP. The largest energy reductions are reported for the JPEG decoder. Additionally, the energy is reduced by an average of 62% for the ARM7 systems. The savings are slightly smaller for the ARM9 and ARM10-based platforms due to the smaller application speedups relative to the ARM7 systems as these are shown in Table 3. More specifically, the system energy is smaller by an average of 59% relative to the all-software solution for the ARM9 systems, while for the ARM10 systems the energy is reduced by 57%. Thus, besides the significant performance improvements derived by adopting the CGC-based data-path, an important reduction in system energy is also reported.

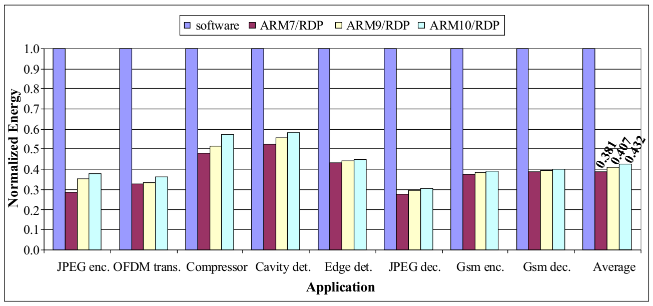

Figure 7. Normalized energy consumption.

The energy-delay product, which is an important design parameter in embedded systems, is presented in Figure 8. In particular, the normalized energy-delay values in respect to the software-only execution of each application are given. It is observed that the ARM/RDP systems achieve considerably better energy-delay product values than the all-microprocessor solution. These improvements in the energy-delay product, ranging from 66% to 92%, are due to the achieved application speedups and due to the system energy reductions relative to the all-microprocessor solutions. The largest reductions are again accomplished for the ARM7 systems as the average decrease is 86%. For the ARM9/RDP platform an average reduction of 83% is reported, while for the ARM10 systems the average reduction is 80%.

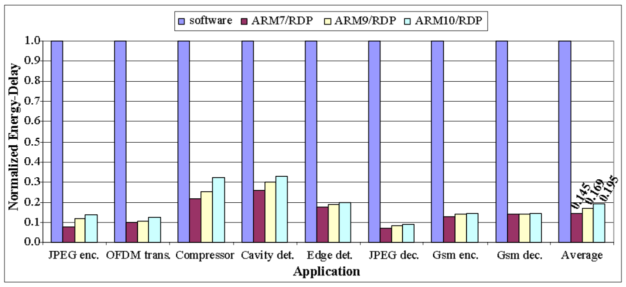

Figure 8. Normalized energy-delay product.

In order to provide an insight into the area cost of coupling a CGC data-path with a microprocessor, we note that the area at 130nm for ARM7 is 2.4mm^2, for ARM9 is 3.2mm^2, while for ARM10 is 6.9mm^2. The synthesized VHDL description of the RDP occupied an area of 0.82mm^2 at 130nm. So, the area of the processing units of the platform increases by an average of 23% for the three ARM-based systems, with a minimum value of 12% for the ARM10 systems and a maximum value of 34% for the ARM7/RDP platforms. Thus, important speedups and energy savings have been achieved with a small area overhead by using a coarse-grained reconfigurable coprocessor.

5. Conclusions

The integration of a Coarse-Grained Reconfigurable Data-Path in a generic single-chip microprocessor system was presented. A design flow was also proposed for mapping complete applications on the system. The ability of the computational resources of the RDP to realize complex arithmetic structures resulted in improving the performance of time critical application segments. Thorough experimentation showed that significant overall speedups, with average value of 2.70, were accomplished. Besides the speedups, important savings in system energy, ranging from 42% to 72%, were reported. Significant energy-delay product reductions are illustrated relative to the software-only executions, while the overhead in system area is small.

6. References

[1] R. Hartenstein, “A Decade of Reconfigurable Computing: A Visionary Retrospective”, in *Proc. of ACM/IEEE DATE ’01*, pp. 642-649, 2001.

[2] S. Hauck et al., “The Chimaera Reconfigurable Functional Unit”, in *IEEE Trans. on VLSI*, vol. 12, no.2, pp. 206-217, Feb. 2004.

[3] Xilinx Inc., Virtex 4 User Guide, September 2005, www.xilinx.com/virtex4.

[4] Altera Corp., Stratix II Device Handbook, July 2005, www.altera.com.

[5] H. Singh et al., “MorphoSys: An Integrated Reconfigurable System for Data-Parallel and Communication-Intensive Applications”, in *IEEE Trans. on Computers*, vol. 49, no. 5, pp. 465-481, May 2000.

[6] J. Becker and A. Thomas, “Scalable Processor Instruction Set Extension”, in *IEEE Design & Test of Computers*, vol. 22, no. 2, pp. 136-148, 2005.

[7] B. Mei et al., “Mapping an H.264/AVC Decoder Onto the ADRES Reconfigurable Architecture”, in *Proc. of FPL ’05*, pp. 622-625, 2005.

[8] S. C. Goldstein et al., “PipeRench: A Coprocessor for Streaming Multimedia Acceleration”, in *Proc. of International Symposium on Computer Architecture* (ISCA), pp. 28-39, 1999.

[9] M. D. Galanis et al., “A Reconfigurable Coarse-Grain Data-Path for Accelerating Computational Intensive Kernels”, in *Journal of Circuits, Systems and Computers* (JCSC), World Scientific Publishers, vol. 14, no. 9, pp. 877-893, August 2005.

[10] M. R. Corazao et al., “Performance Optimization Using Template Mapping for Datapath-Intensive High-Level Synthesis”, in *IEEE Trans. on CAD*, vol.15, no. 2, pp. 877-888, August 1996.

[11] N. Cheung et al., “INSIDE: Instruction Selection/Identification & Design Exploration for Extensible Processors”, in *IEEE/ACM ICCAD ‘03*, pp. 291-297, 2003.

[12] J. Cong et al., “Application-Specific Instruction Generation for Configurable Processor Architectures”, in *Proc. of the ACM FPGA*, pp. 183-189, 2004.

[13] R. Kastner et al., “Instruction Generation for Hybrid Reconfigurable Systems”, in *ACM TODAES*, vol. 7, no.4, pp. 605-627, October 2002.

[14] G. Venkataramani et al., “Automatic Compilation to a Coarse-Grained Reconfigurable System-on-Chip”, in *ACM TECS*, vol. 2, no. 4, pp. 560-589, Nov. 2003.

[15] Y. Kim et al., “Design and Evaluation of a Coarse-Grained Reconfigurable Architecture”, in *Proc. of ISOCC ’04*, pp. 227-230, 2004.

[16] B. Mei, et al., “Mapping methodology for a Tightly Coupled VLIW/Reconfigurable Matrix Architecture, A Case Study”, in *Proc. of ACM/IEEE DATE ’04*, pp. 1224-1229, 2004.

[17] F. Barat et al., “Low Power Coarse-Grained Reconfigurable Instruction Set Processor”, in *Proc. of FPL ’03*, LNCS 2778, Springer, pp. 230-239, 2003.

[18] J. W. Crenshaw, “MATH Toolkit for Real-Time Programming”, *CMP Books*, 2000.

[19] ARM Corp., www.arm.com, 2006.

[20] D. G. Chinnery and K. Keutzer, “Closing the Power Gap between ASIC and Custom: An ASIC Perspective”, in *Proc. of Design Automation Conference* (DAC), pp. 275-280, 2005.

[21] Intel StrongARM 1110 processor, www.intel.com.

[22] Artisan Components, www.artisan.com.