Dynamic Hardware Reconfigurations: Performance Impact for MPEG2

Elena Moscu Panainte, Koen Bertels, and Stamatis Vassiliadis

Computer Engineering Lab
Delft University of Technology, The Netherlands
{E.Panainte, K.Bertels, S.Vassiliadis}@et.tudelft.nl

Abstract. In this paper, we study the impact dynamic reconfiguration has on the performance of current reconfigurable technology. As a testbed, we use the Xilinx Virtex II Pro, the Molen experimental platform and the MPEG2 encoder as the application. We show for the MPEG2 encoder that a substantial overall performance improvement, up to 34 %, can be achieved when SAD, DCT and IDCT functions are executed on the reconfigurable hardware when the compiler anticipates and separates configuration from execution. This study also considers the impact inappropriate scheduling can have on the overall performance. We show that slowdowns of up to a factor 1000 are observed when the configuration latency is not hidden by the compiler. Our experiments show that appropriate scheduling allows to exploit up to 97% of the maximal theoretical speedup.

1 Introduction

The development of architectural improvements is a complex process as it deals with a large number of highly interconnected factors. An improvement in one component does not necessarily result in an improved system performance. This complexity increases considerably as heterogeneous architectures are included. The combination of a general purpose processor (GPP) and a Field Programmable Gate Array (FPGA) is becoming increasingly popular (e.g. [1], [2], [3]) as it allows developers to better partition and manage their projects (e.g. [4], [5] and [6]). Reconfigurable computing is a new style of computer architecture which, thanks to the availability of high density programmable logic chips, allows the designer to combine the advantages of both hardware (speed) and software (flexibility). A general paradigm that eliminates the shortcomings of other approaches in reconfigurable computing is described in [7] and in [8] and is referred to as the Molen Programming Paradigm. An important drawback of RC paradigm is the huge reconfiguration latency of the actual FPGA platforms. Based on the work described in [9] where a compiler for the Molen approach was presented, this paper addresses some open issues which primarily involve the hardware reconfiguration impact on performance. As will be explained in the remainder of this paper, the potential speedup of the kernel hardware executions can be completely wasted by inappropriate repetitive hardware reconfigurations.

In this paper, we investigate the impact on the overall performance for the MPEG 2 benchmark of hardware reconfiguration in two cases: a) the straightforward approach when each hardware execution is preceded by the coresponding hardware configuration and b) when the hardware configuration can be anticipated and efficiently scheduled referring to the hardware execution. In this paper, we only Analise MPEG 2 encoder benchmark. Whenever in the remainder of the paper, we mention MPEG 2, we refer only to MPEG 2 encoding phase.

The main contributions of the paper can be summarized as follows:

- Based on profiling results, we determine that the maximal performance improvement of the Molen approach versus the pure software approach for the MPEG 2 benchmark that can be achieved by hardware execution of the kernel operations SAD, DCT and IDCT is about 65 %. We consider a set of real hardware implementations of these kernels and determine that the kernels hardware execution is up to 31x faster than the pure software execution.
- We estimate that, in the straightforward approach when each hardware execution is preceded by the coresponding hardware configuration, the huge reconfiguration latency of the hardware reconfiguration can slowdown the MPEG 2 benchmark by 3 order of magnitudes.
- A scheduling that anticipates the hardware configuration can eliminate the previous described drawback and provide a performance improvement up to 97 % from the maximal performance improvement of MPEG 2 benchmark.

The paper is organized as follows: in the next section, we present the Molen programming paradigm and describe a particular implementation, called the Molen processor. Section 3 describes the necessary compiler extensions for the Molen programming paradigm. Consequently, we present a profiling experiment and analyze the impact on performance of the hardware reconfiguration for the MPEG 2 benchmark. Finally, we conclude by discussing future research directions.

2 The Molen Programming Paradigm

The Molen programming paradigm [8] is a sequential consistency paradigm for programming CCMs possibly including a general purpose computational engine(s). The paradigm allows for parallel and concurrent hardware execution and is intended (currently) for single program execution. It requires only a one time architectural extension of few instructions to provide a large user reconfigurable operation space. The added instructions include **SET** $<$ *address* $>$ implying that at a particular location the hardware configuration logic is defined and **EXECUTE** $<$ *address* $>$ that serves to control the executions of the operations on the reconfigurable hardware. In addition, two MOVE instructions for passing values to and from the GPP register file and the reconfigurable hardware are required.

For the moment, we only consider code fragments in the form of functions having a number of parameters. These parameters are passed to special reconfigurable hardware registers denoted as Exchange Registers (XRs). In order to

maintain the correct program semantics, the code is annotated and Custom Computing Machine (CCM) description files provide the compiler with implementation specific information such as the addresses where the SET and EXECUTE code are to be stored, the number of exchange registers, etc. It should be noted that this programming paradigm allows modularity, meaning that if the interfaces to the compiler are respected and if the instruction set extension (as described above) is supported, then custom computing hardware provided by multiple vendors can be incorporated by the compiler for the execution of the same application. The modular approach also implies that the application can be ported to multiple platforms with mere recompilation.

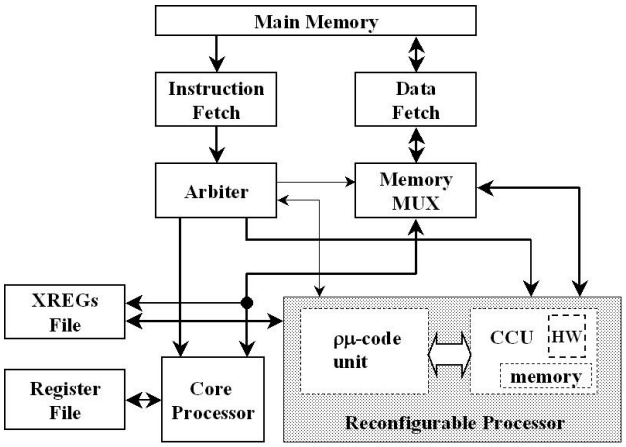

Fig. 1. The Molen machine organization

Finally, it is noted that every user is provided with at least $2^{(n-op)}$ directly addressable functions, where n represents the instruction length and 'op' the opcode length. The number of addressable functions can be easily augmented to an arbitrary number by reserving additional opcodes for indirect opcode accessing.

The Molen reconfigurable processor: The Molen $\rho\mu$-coded processor has been designed having in mind the programming paradigm previously presented. The Molen machine organization is depicted in Figure 1. The arbiter performs a partial decoding of the instructions fetched from the main memory and issues them to the corresponding execution unit. The parameters for the FPGA reside in the Exchange Registers. In the Molen approach, an extended microcode - named reconfigurable microcode - is used for the emulation of both SET and EXECUTE instructions. The microcode is generated when the hardware implementation for a specific operation is designed and it cannot be further modified.

3 Compiler Extensions for the Molen Programming Paradigm

The compiler system relies on the Stanford SUIF2[10] (Stanford University Intermediate Format) Compiler Infrastructure for the front-end, while the back-end is built over the framework offered by the Harvard Machine SUIF[11]. The last component has been designed with retargetability in mind. It provides a set of back-ends for GPPs, powerful optimizations, transformations and analysis passes. These are essential features for a compiler targeting a CCM. We have currently implemented the following extensions for the x86 processor:

- A special pass in the SUIF front-end identifies the code that is mapped on the reconfigurable hardware. Using special pragma annotation, all the calls of those functions are marked for further modification.
- The Instruction Set has been extended with SET/ EXECUTE instructions at both MIR (Medium Intermediate Representation) level and LIR (Low Intermediate Representation) level.
- Exchange Registers (XRs) are added to the Register File Set. These registers are used for passing operation parameters to the reconfigurable hardware and returning the computed values after the operation execution. In order to avoid dependencies between the RU and GPP, the XRs receive their data directly from the GPP registers. The XR allocation phase, introduced in Machine SUIF at LIR level, precedes the GPP register allocation. The conventions introduced for the XRs are implemented in this pass.
- Code generation for the reconfigurable hardware is performed when translating SUIF to Machine SUIF IR, and affects the function calls marked in the front-end.

An example of the code generated by the extended compiler for a function call when the considered function is executed on the reconfigurable fabric is presented in Figure 2. The standard function call is replaced by the appropriate instructions for sending parameters (two for the considered example) to the reconfigurable hardware in XRs, hardware configuration phase, hardware execution and finally returning the computed result to the GPP registers. The presented code is at Medium-level Intermediate Representation (MIR), before register allocation and code finalization passes.

```
mov    $vr2.s32 ← param1
movtx  $vr1.s32(XR) ← $vr2.s32 # send param1 in XR
mov    $vr4.s32 ← param2
movtx  $vr3.s32(XR) ← $vr4.s32 # send param2 in XR
set    address_op1_SET          # hardware configuration
exec   address_op2_EXEC         # hardware execution
movfx  $vr6.s32 ← $vr5.s32(XR) # return result
mov    res ← $vr6.s32
```

Fig. 2. MIR Code generated by the Molen compiler

Certain information about the target architecture such as microcode address of SET and EXECUTE instructions, the number of XRs, the fixed XR associated with each operation, etc. are extracted by the compiler from a description file.

4 A MultiMedia Based Evaluation

In order to evaluate the impact on performance of the hardware configuration we consider the MPEG2 encoder multimedia benchmarks and the test sequences presented in Table 1. Building on previous work [9][7], we look at the following time consuming functions that are implemented in reconfigurable hardware: SAD (sum of absolute-difference), 2D DCT (2 dimensional discrete cosine transform) and IDCT (inverse DCT). As explained before, we consider a Molen machine organization with an x86 as the Core Processor. More specifically, the compiler generates code for the x86 architecture while the measurements are performed on an AMD Athlon XP 1900+ at 1600 MHz.

Name	# frames	Resolution
carphone	96	176x144
claire	168	360x288
container	300	352x288
football	125	352x240
foreman	300	352x288
garden	115	352x240
mobile	140	352x240
tennis	112	352x240

Table 1. MPEG test sequences

MPEG 2 Profiling Results for Pure Software Execution We first compute the number of cycles each function consumes for the input sequences given in Table 1, when executed on the target GPP without reconfigurable hardware acceleration. These profiling results for the MPEG2 encoder benchmarks are presented in Table 2. The cumulated time spent by SAD, DCT and IDCT functions (Table 2, column 3,5 and 7) in the pure software approach represents about 65 % of the total MPEG2 execution time. In consequence, the hardware acceleration of these functions (as proposed in the Molen approach) can produce a significant speedup of the MPEG2 encoder up to 3x. The results from Table 2, column 3 suggest that the SAD function is the best candidate for hardware implementation as it can provide up to around 40 % performance improvement. Whereas for the encoding phase, IDCT cannot yield substantial performance improvement, in decoding, this function is heavily used and can produce a significant performance increase.

MPEG2 Performance Estimation for Molen CCM Execution As the presented Molen CCM is not currently implemented, we determine the perfor-

Video	SAD (16x16)		DCT (8x8)		IDCT (8x8)	
sequence	# Cycles	% Time	# Cycles	% Time	# Cycles	% Time
carphone	997	31.69	37796	28.19	2612	1.95
claire	1092	36.46	37796	26.44	2177	1.53
container	1008	34.44	37590	27.04	2208	1.59
football	1484	42.74	37537	22.93	2827	1.73
foreman	1298	39.93	37572	24.35	2193	1.42
garden	1311	40.21	37594	24.70	2463	1.62
mobile	1092	35.95	37536	26.30	2519	1.77
tennis	1344	41.23	37531	24.39	2221	1.44
Average	1203	37.83	37593	25.54	2402	1.63

Table 2. Profiling results for MPEG2 encoder

mance of the Molen CCM based on the measured profiling results for the GPP included in the MOLEN CCM as follows:

$$n_{Molen} \simeq n_{X86} - n_f + n_{call} \cdot cost \tag{1}$$

$$cost = x_{SET} + y_{EXEC} \tag{2}$$

where

- n_{Molen}: the total number of GPP cycles spent in the considered application by the Molen processor;
- n_{X86}: the total number of GPP cycles when the considered
- function f is implemented on the FPGA application is executed exclusively on the GPP;
- n_f: the total number of GPP cycles spent in function f when the considered application is performed only on the GPP;
- n_{call}: the number of calls to function f in the considered application;
- $cost$: the number of cycles for one execution of function f on FPGA;
- x_{SET}: the number of GPP cycles required for one configuration of the FPGA for function f;
- y_{EXEC}: the number of GPP cycles required for one execution on the FPGA of function f; for the considered hardware implementation, the execution time is not dependent on the input data.

In our experiments, we have measured the values for n_{X86}, n_f and n_{call} included in Formula 1. To this purpose, we used the *Halt* library[12] available in Machine SUIF. This library is an instrumentation package that allows the compiler to change the code of the program being compiled in order to collect information about the program own behavior (at run-time). In order to minimize the impact of external factors on the measurements, we run the applications in single mode and with the highest priority in Linux.

Hardware Execution and Reconfiguration Before discussing the hardware acceleration, we present the target FPGA platform included in our experiments. We used the Xilinx Virtex II Pro, XC2VP20 chip and the 2D DCT and

2D IDCT cores available as IPs in the Xilinx Core Generator Tool as well as the SAD implementation presented in [7]. After synthesis, the area required by each function is given in Table 3, column 3. We measured the hardware execution time of each function in terms of the target Athlon processor cycles, given in Table 3, column 3. Based on the characteristics of the XC2VP20 chip, for which a complete configuration of 9280 slices takes about 20 ms, we estimate the reconfiguration time for the considered functions as presented in Table 3, column 2.

Op	Area	EXEC	HW Speedup	SET		SET_MAX		SET/SET_MAX
	[slices]	[cycles]		[ms]	[cycles]	Mean	st.dev	
SAD	831	133	9 x	2	3200000	1070	167	2991
DCT	4314	1184	31 x	10	16000000	36409	80	439
IDCT	5436	1200	2 x	12	19200000	1202	225	15973

Table 3. Hardware configuration and execution parameters

We basically performed two experiments to assess not only the impact of hardware acceleration but also the impact of an appropriate scheduling of the reconfiguration phase.

A Simple Hardware Reconfiguration Scheduling On the basis of the hardware execution times from Table 3 and the average software execution time given in Table 2 column 2,4,6, we determine that the hardware acceleration of the considered kernels (Table 3, column 4) is up to 31x. However, a direct scheduling where the corresponding SET and EXECUTE instructions for hardware configuration and hardware execution are consecutively executed for each operation can completely waste the hardware speedup. In this consecutive scheduling, the hardware reconfiguration is each time performed before the hardware execution. Due to the huge reconfiguration latency and repetitive hardware configuration imposed by this scheduling, the use of reconfigurable hardware can slowdown the MPEG2 benchmark (computed as n_{Molen}/n_{X86} using Formula 1) by 2-3 orders of magnitude (Table 4, row 2) compared to complete execution on the GPP alone.

Based on the profiling result (Table 2), reconfigurable hardware execution times (Table 3) and Formula 1, we determine the upper boundary for a SET instruction latency that ensures that the Molen CCM is not slower than the pure software approach ($n_{Molen} \simeq n_{X86}$). We refer to this boundary as SET_MAX and is described by:

$$SET\_MAX \simeq \frac{n_f}{n_{call}} - y_{EXEC} \tag{3}$$

The mean SET_MAX values and standard deviations are presented in Table 3, (columns 7-8). We notice that the complete hardware configuration of the currently available FPGA platforms (SET) accounts for 3-4 orders of magnitude (see Table 3, column 9) more reconfiguration time than SET_MAX and produces for the MPEG 2 benchmark a performance decreasing of 2-3 order of magnitude

(Table 3, last column). In consequence, without an appropriate scheduling of the SET instructions, the overall performance is decreased due to the huge reconfiguration latency in spite of the faster hardware execution time. Such an appropriate scheduling is discussed in the rest of this section.

Out of Loop Hardware Reconfiguration The above presented limitation can be eliminated by simply scheduling the hardware configuration phase as early as possible. This transformation can be particularly beneficial when there is only one operation executed in hardware included in a loop-body. This situation is encountered in MPEG2 encoder for all three considered functions. In the rest of this section, we estimate the effect of this transformation on the performance of the Molen processor. In this respect, we use the ceteris paribus approach meaning that we look at the influence of each function individually to estimate the performance improvement while considering that none of the other functions are implemented in reconfigurable hardware. As we previously explained, performing the hardware configuration before each hardware execution can decrease the performance for the Molen processor versus the GPP alone. Nevertheless, removing the unnecessary repetitive SET instructions (when the hardware is already configured) results in a significant performance improvement (computed as $\frac{n_f - n_{call} * y_{EXEC} - x_{SET}}{n_{X86}}$). The performance efficiency (presented in Table 4, row 4) emphasizes that the individual improvement of each function is very close to the maximum possible improvement.

Video sequence	SAD	DCT	IDCT
Simple Scheduling Slowdown	1012 x	108 x	131 x
Out-of-loop Scheduling Performance Impr	33.61 %	24.68 %	0.75 %
Theor. Maximal Performance Impr	37.83 %	25.54 %	1.63 %
Performance Efficiency	89 %	97 %	46 %

Table 4. MPEG 2 encoder performance results with and without anticipated hardware configuration

We emphasize that the execution of both SAD and DCT simultaneously on the reconfigurable hardware will not provide a cumulative performance improvement due to the required switching of configurations in order to preserve the overall application behavior. The compiler optimizations are expected to play a key role in handling these more complicated cases.

5 Conclusions

In this paper, we used the compiler technology developed to support the Molen programming paradigm to study the conditions under which substantial perfor-

mance improvements can be obtained with hardware acceleration using CCM's. Based on profiling results, we showed that potential speedups can be completely outweighed by inappropriate scheduling of the reconfiguration instruction. When theoretically a performance improvement of up to 40 % is achievable, the slowdown caused by improper scheduling can be as large as a factor 1000 (e.g. for SAD). We also showed that given a suitable scheduling up to 97 % of the maximal performance improvement can be obtained.

Future research will focus on compiler optimizations to allow for concurrent execution. We also intend to extend the compiler to take into account complex knowledge about the target reconfigurable platform and thus to achieve an efficient schedule of the different operations performed on the reconfigurable fabric.

References

1. Campi, F., Toma, M., Lodi, A., Cappelli, A., Canegallo, R., Guerrieri, R.: A VLIW Processor with Reconfigurable Instruction Set for Embedded Applications. In: In ISSCC Digest of Technical Papers. (2003) 250–251
2. Sima, M., Vassiliadis, S., S.Cotofana, van Eijndhoven, J., Vissers, K.: Field-Programmable Custom Computing Machines - A Taxonomy. In: 12th International Conference on Field Programmable Logic and Applications (FPL). Volume 2438., Montpellier, France, Springer-Verlag Lecture Notes in Computer Science (LNCS) (2002) 79–88
3. Becker, J.: Configurable Systems-on-Chip : Commercial and Academic Approaches. In: Proc. of 9th IEEE Int. Conf. on Electronic Circuits and Systems - ICECS 2002, Dubrovnik, Croatia (2002) 809–812
4. Gokhale, M.B., Stone, J.M.: Napa C: Compiling for a Hybrid RISC/FPGA Architecture. In: Proceedings of FCCM'98, Napa Valley, CA (1998) 126–137
5. Rosa, A.L., Lavagno, L., Passerone, C.: Hardware/Software Design Space Exploration for a Reconfigurable Processor. In: Proc. of DATE 2003, Munich, Germany (2003) 570–575
6. Ye, Z.A., Shenoy, N., Banerjee, P.: A C Compiler for a Processor with a Reconfigurable Functional Unit. In: ACM/SIGDA Symposium on FPGAs, Monterey, California, USA (2000) 95–100
7. Vassiliadis, S., Wong, S., Cotofana, S.: The MOLEN $\rho\mu$-Coded Processor. In: 11th International Conference on Field Programmable Logic and Applications (FPL). Volume 2147., Belfast, UK, Springer-Verlag Lecture Notes in Computer Science (LNCS) (2001) 275–285
8. Vassiliadis, S., Gaydadjiev, G., Bertels, K., Moscu Panainte, E.: The Molen Programming Paradigm. In: Proceedings of the Third International Workshop on Systems, Architectures, Modeling, and Simulation, Samos, Greece (2003) 1–7
9. Moscu Panainte, E., Bertels, K., Vassiliadis, S.: Compiling for the Molen Programming Paradigm. In: 13th International Conference on Field Programmable Logic and Applications (FPL). Volume 2778., Lisbon, Portugal, Springer-Verlag Lecture Notes in Computer Science (LNCS) (2003) 900–910
10. (http://suif.stanford.edu/suif/suif2)
11. (http://www.eecs.hardvard.edu/hube/research/machsuif.html)
12. M.Mercaldi, Smith, M.D., Holloway, G.: The Halt Library. In: The Machine-SUIF Documentation Set, Hardvard University (2002)