MULTIMEDIA RECONFIGURABLE HARDWARE DESIGN SPACE EXPLORATION

Elena Moscu Panainte, Koen Bertels, Stamatis Vassiliadis
Computer Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
TU Delft, The Netherlands
{elena, koen, stamatis}@ce.et.tudelft.nl

ABSTRACT
In this paper we consider a set of multimedia applications and investigate the potential performance impact a reconfigurable microcoded processor can provide when added to a general purpose core processor. In a design space exploration, considering MPEG2 and JPEG benchmarks, we investigate performance boundaries, memory bottlenecks and the influence the core and reconfigurable processor communication has on performance. Under some realistic scenarios and serial FPGA execution, it is shown that a 53 % cycle reduction is expected when comparing a design having a core processor and a design when the core processor is augmented with a reconfigurable microcoded engine. In addition, we have found that transferring parameters between the core processor and the reconfigurable processor may not severely influence the overall performance. Finally we investigated the memory bandwidth for operations mapped automatically on FPGA. The case study indicates that small latency DCT hardware design performs well when interfaced with 512 bytes/cycle. Our studies also indicate that about 64 bytes/cycle will support high speed execution for SAD and IDCT.

KEYWORDS
Design space exploration, reconfigurable architecture, multimedia application, Molen programming paradigm

1 Introduction

As reconfigurable architectures (e.g. [1], [2]) come to age programming, compilation and design space explorations become increasingly important. Numerous approaches have been developed, including [3] and [4], to address such demands. An approach of reference for Reconfigurable Computing (RC) includes the Molen machine organization [1] with a prototype implementation on Virtex II Pro [5], the Molen programming paradigm [6] and a C compiler [7] that incorporates the frequent programming and architectural Molen reconfigurable features. In [7] it has been reported that the Molen programming paradigm and RC can substantially reduce the number of instructions executed in some multimedia applications. In this paper we examine the potential of the Molen approach in terms of execution time for the well-known multimedia applications MPEG2 and JPEG encoders and decoders. The multimedia benchmarks are particularly suitable for the Molen approach as they usually involve intensive computation for highly regular operations, intensive I/O or memory accesses and require real-time processing capabilities. More specifically, we perform a design space exploration study and quantitatively analyze:

- **performance boundaries**: we first determine the maximal performance gains for each operation implemented on the reconfigurable hardware. We also compute for each operation the latency range of the valid hardware designs whose execution on the reconfigurable hardware is faster then the pure software execution. Consequently we show that for real operation implementations the MPEG2 encoder executed on the Molen processor achieves a 53 % reduction of cycles of total execution time.
- **parameter exchange**: we investigate the effects on performance of the parameter passing between the general purpose processor (GPP) and reconfigurable hardware and we show that the overhead is negligible.
- **memory bottlenecks**: we examine the effect of the data communication between the reconfigurable hardware and memory on performance and show that for DCT a high IO bandwidth (512 bytes/cycle) is required when a fast execution time of around 20-30 cycles is imposed. For SAD and IDCT, the data communication bandwidth is not a constraint.

On the basis of our design space exploration, the hardware designer can compute in advance for each hardware implementation the global performance improvement and the influence of memory or parameter passing latencies on the overall performance. For example, when a specific speed-up is imposed, the designer is aided to choose the operations that can achieve the required speed-up and the IO bandwidth that eliminates the bottlenecks in the system.

The existing literature for multimedia applications analyzes the achieved improvement when only one operation is implemented in reconfigurable hardware (IDCT in MPEG decoder is presented [8]) or presents the profile information for a set of operations, but relying on a small number of test sequences (two sequences in [9]), which is

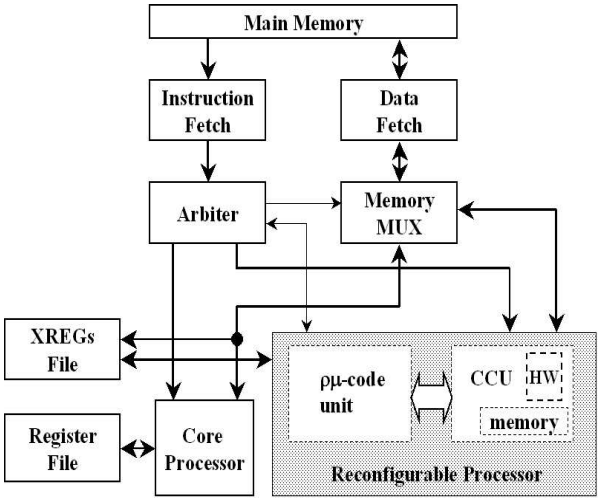

Figure 1. The Molen machine organization

insufficient because it is generally known that the behavior of specific operations (such as VLC) is highly dependent on the characteristics of the input data. In [10] it is presented an analysis of the overall characteristics of the multimedia applications, such as operation frequencies, basic block and branch statistics. Our approach is different as we analyze four multimedia applications for a large set of representative input data and we focus on the study of a set of specific operations in the context of the overall application. Moreover, we provide and analyze profiling information which are specifically relevant for the Molen processor, such as the execution time, hardware configuration and communication overhead between the processor, reconfigurable hardware and memory.

The discussion of the paper is organized as follows. In Section 2 the Molen programming paradigm and compiler are briefly introduced. In the following section we present the results of a multimedia case study where a number of computation-intensive operations from a set of multimedia benchmarks using a large number of input data are analyzed. Finally, conclusions and future work are drawn in Section 4.

2 Background

As in the design space exploration presented in the next section the Molen approach is assumed, we provide a brief introduction of the Molen machine organization ([1]), the Molen programming paradigm ([6]) and the compilation tools [7].

Molen Machine Organization: The main Molen components as depicted in Figure 1 are: the Core Processor - which is a GPP, and the *Reconfigurable Processor* - implemented in the FPGA. The Arbiter performs a partial decoding of the instructions fetched from main memory and issues them to the GPP or the reconfigurable processor. The division in hardware and software part is directly mappable to the two processors. The hardware targeted pieces are executed by the FPGA, while the (remaining) software modules are executed on the GPP.

Name	# frames	Resolution
carphone	96	176x144
claire	168	360x288
container	300	352x288
football	125	352x240
foreman	300	352x288
garden	115	352x240
mobile	140	352x240
standard	3	128x128
tennis	112	352x240

Table 1. MPEG test sequences in YUV format

The Molen Programming Paradigm The Molen programming paradigm [6] is a sequential consistency paradigm for programming Field-programmable Custom Computing Machines (FCCMs). For any given instruction set architecture an one-time architectural extension of few instructions is sufficient to provide a large user reconfigurable operation space. The minimal extension comprises 4 instructions (for the minimal πISA as defined in [6]): two instructions (**set/execute**) for loading a hardware implementation and launching its execution on the reconfigurable hardware and two instructions (**movtx/movfx**) for the communication between the reconfigurable hardware and GPP.

Compilation Tools: The compiler [7] relies on the Stanford SUIF2 (Stanford University Intermediate Format) and the Harvard Machine SUIF back-end framework. The x86 back-end already available in Machine SUIF was extended to generate code for the Molen FCCM with an x86 processor as GPP. The compiler automatically maps the application on the target FCCM based on pragma annotations. It is also used to generate additional code that gathers profile information for the compiled application at the execution time.

3 The MPEG2 and JPEG Case Study

In this section we explore the hardware constraints for implementing on an FPGA a set of well-known time-consuming multimedia operations. The main goal is to determine the parameters that have a substantial impact on the system performance and their range of values in order for the Molen processor to outperform the standalone GPP.

Target Architecture and Applications 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. The considered applications are a set of multimedia benchmarks consisting of the Berkeley MPEG2 encoder and decoder and the SPEC95 JPEG encoder and decoder.

Name	Resolution
boat.ppm	512x512
clegg.ppm	814x880
frymire.ppm	1118x1105
lena.ppm	512x512
mandrill.ppm	507x509
monarch.ppm	768x512
peppers.ppm	512x512
sail.ppm	768x512
serrano.ppm	629x794
tulips.ppm	768x512
penguin.ppm	1024x739
specmun.ppm	1024x768
vigo.ppm	1024x768

Table 2. JPEG test images

The time-consuming operations candidate for hardware execution are SAD (sum of absolute-difference), 2D DCT (2 dimensional discrete cosine transform), IDCT (inverse DCT), VLC (variable length coding) and VLD (variable length decoding). The input data are representative series of test images and scenes of various sizes, presented in Tables 1 and 2.

In this paper we assume that the GPP and FPGA do not run concurrently and that the execution of an operation on the FPGA is each time preceded by the FPGA configuration (even if the previous configuration is the same). Moreover, we assume that the FPGA performs only one operation at the same time. In order to evaluate the performance of the Molen processor for one application where function f is executed on the FPGA, we compute the number of GPP cycles for the Molen processor using:

$$n_{Molen} \simeq n_{X86} - n_f + n_{call} \cdot cost \quad (1)$$

$$cost = x_{SET} + y_{EXEC} + n_{par} * z_{MOV\_XR} + c$$

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 application is executed exclusively on the GPP;
- n_f: the total number of GPP cycles spent in all executions of function f on the GPP;
- n_{call}: the number of calls to function f in the considered application;
- $cost$: the number of GPP cycles for one execution of function f on FPGA; the time for FPGA configuration and execution is converted in GPP cycles.
- 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; it may depend on the input data. In order to be constant for the chosen set of input data we consider the largest values;
- n_{par}: the number of instructions for sending the parameters from GPR to XR and returning the results;
- z_{MOV_XR}: the number of GPP cycles for one MOV_XR instruction (movtx or movfx)
- c: quantifies the the calling convention differences in number of GPP cycles. As c is small (< 10 cycles) for the considered applications, we neglect it.

In our design space exploration, we first analyze the pure software execution and extract the relevant profile information for the considered applications and functions. Based on the profile information and Formula 1, we examine the performance and the hardware parameters for the target Molen FCCM. The profile information is extracted using *Halt Library* [11] for code instrumentation. Additionally, we develop a set of analysis routines to measure the number of cycles executed in a specific function (using RDTSC - Read Time Stamp Counter instruction) and the number of function calls. More specifically, we have measured the values for n_{X86}, n_f and n_{call} included in Formula 1. In order to minimize the impact of extern factors on the measurements, we run the applications in single mode and with the highest priority in Linux.

As illustrated in Formula 1, the cost per function call for a reconfigurable execution is determined by the cost of the FPGA configuration (x_{SET}), FPGA execution (y_{EXEC}) and transfer of parameters (z_{MOV_XR}). The influence of these factors on the overall performance and their optimal ranges are explored in the rest of this section.

	JPEG encoder		JPEG decoder	
Input	VLC	DCT	IDCT	VLD
boat	**2272**	**2746**	**2905**	**7322**
clegg	3631	2748	3701	13950
frymire	3371	2758	3313	12989
lena	2469	2759	3461	8080
mandrill	3463	2759	3686	12967
monarch	2403	2758	3360	7836
penguin	2488	2762	3453	8014
peppers	2505	2764	3463	8369
sail	3110	2758	3545	11276
serrano	2955	2766	3698	10753
specmun	2375	2776	3389	7389
tulips	2882	2760	3571	10177
vigo	2550	2755	3424	8444

Table 4. Software cost (bold) expressed in GPP cycles for the functions included in JPEG application

Cost Range The purpose of the GPP extension with reconfigurable hardware is to achieve a performance improvement over the GPP alone, meaning $n_{Molen} < n_{X86}$

	MPEG encoder					MPEG decoder		
Input	SAD	DCT	IDCT	VLC I	VLC II	IDCT	VLD I	VLD II
carphone	**997**	37796	2612	2631	2196	2513	1763	1347
claire	1092	37595	**2177**	1710	1524	**2056**	745	659
container	1008	37590	2208	1842	**1476**	2087	880	**586**
football	1484	37537	2827	2795	2318	2678	1940	1499
foreman	1298	37572	2193	**1577**	1494	2071	**568**	606
garden	1311	37594	2463	2046	1524	2332	1091	662
mobile	1092	37536	2519	2123	1564	2398	1177	722
standard	1199	37549	3423	2930	2239	3295	-	-
tennis	1344	**37531**	2221	1702	1578	2099	718	713

Table 3. Software cost (bold) expressed in GPP cycles for the functions included in MPEG2 application

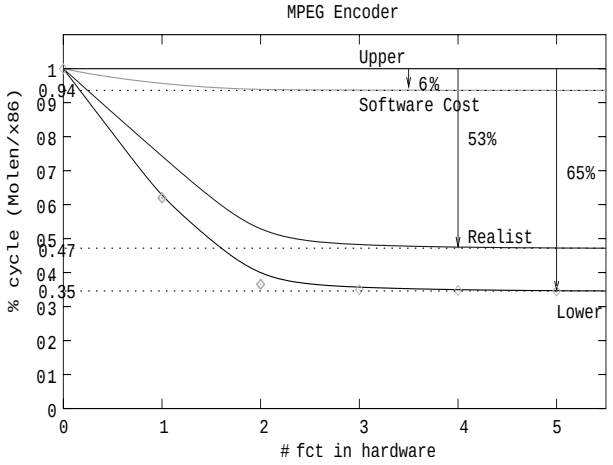

Figure 2. Relative performance boundaries and a real implementation analysis

which holds when

$$cost < n_f/n_{call}. \quad (2)$$

The values for limit cost (n_f/n_{call}) in Formula 2 are presented in Tables 3 and 4. They represent the cut-off points for the hardware execution from where an implementation provides a performance improvement. We refer to the minimal values of each operation as the software cost (presented in bold in Tables 3 and 4). For an implementation that executes the operation in a number of cycles less than the software cost, a performance improvement is guaranteed to hold for all input data in the study.

Performance boundaries For each operation, we determine the number of cycles it consumes in the pure software approach from the overall application (n_f/n_{X86}) as presented in Table 5 (second column). These values represent the maximal improvements of the overall performance that can be achieved by hardware acceleration of the considered functions. We notice that implementing the SAD function on the FPGA can improve the overall performance up to 38 % while the overall improvement for VLC is very low (0.2 %). When taking into account all the functions for MPEG2 encoder (Fig. 2), the maximal reduction of the number of cycles is 65 % compared to the pure software implementation.

We are also interested in determining the boundaries between which any real implementation should be situated. The upper boundary is when there is no improvement, meaning $n_{Molen} = n_{X86}$ and the lower (theoretical) boundary corresponds to an infinite hardware acceleration ($cost$ = 0) of each function. These boundaries are presented in Fig. 2 and they limit the design space where a hardware designer should place a particular implementation. When all operations are designed to execute at the software cost, then an overall performance improvement is still guaranteed (6 % in Fig. 2). This improvement is due to the safe choice of minimal value for the software cost in order to guarantee no performance decreasing even for the worst case input data.

For the real, non-optimized FPGA implementations described in [1], we also plotted in Fig. 2 the performance for the same operations assuming that compiler optimizations hide the configuration latency ($x_{SET} = 0$). After converting the reported number of cycles to our target processor, we obtain 53% performance improvement.

Parameter Passing Impact In order to understand the impact of the Molen parameter passing mechanism, we assume a scenario in which the cost from Formula 1 is exclusively spent for passing parameters ($cost = n_{par} * z_{MOV\_XR}$ and $x_{SET} = y_{EXEC} = 0$). Under this theoretical assumption we compute the maximal number of cycles for z_{MOV_XR} as $z_{MOV\_XR} = software\_cost/n_{par}$ given in Table 5 (last column). In order to interpret the results, it is important to realize that MOV_XR instructions resemble the move general purpose register instructions which usually require a small number (~ 3) of cycles. Our computations show that for the SAD function, a MOV_XR instruction can be executed in up to 166 cycles before the maximal performance (38 % in Table 5, second column) is consumed. In the case of DCT, the MOV_XR can take up to 37531 cycles before the penalty is higher than the maximal performance gain of 25.4 %.

In order to analyze the communication overhead between GPP and FPGA, we assumed an exaggerated sce-

Function	MAX Improv	MOV_XR max
MPEG2 encoder		
SAD	38.0 %	166
DCT	25.4 %	37531
IDCT	1.6 %	2177
VLC I	0.2 %	225
VLC II	0.1 %	369
MPEG2 decoder		
IDCT	38.3 %	2056
VLD I	3.1 %	284
VLD II	2.0 %	586
JPEG encoder		
VLC	14.7 %	568
DCT	14.0 %	2746
JPEG decoder		
IDCT	49.5 %	581
VLD	24.3 %	732

Table 5. Marginal improvement for each function and the maximal cost for MOV_XR (cycles)

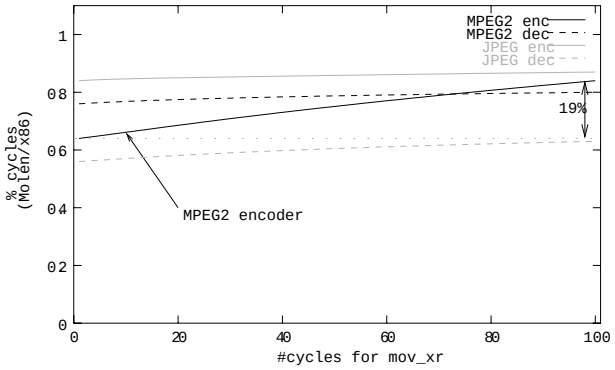

Figure 3. z_{MOV_XR} impact when passing parameters requires software cost/2

nario in which the cost for hardware configuration and execution is half of the software cost. The impact of z_{MOV_XR} is presented in Fig. 3 showing that the MPEG2 encoder is the only application whose performance may be negatively influenced by z_{MOV_XR}. This is explained by the low software cost for SAD (compared to DCT, Table 3) and the large number (8) of parameters. In conclusion, we consider that for the operations under considerations, transferring parameters and returning the results is not a bottleneck in the system.

Communication FPGA - Memory Finally, we investigate the FPGA-memory data communication bandwidth as some parameters passed to the FPGA in XRs are pointers to blocks of data placed in external memory. In this context, we assume that access to memory is sequential and symmetrical (number of cycles to read and write one block of data are equal). In order to determine the amount of data transferred to/from external memory, we introduce a special pass in the compiler to annotate each basic block of the considered functions with the number of read and write memory instructions, as well as the corresponding number

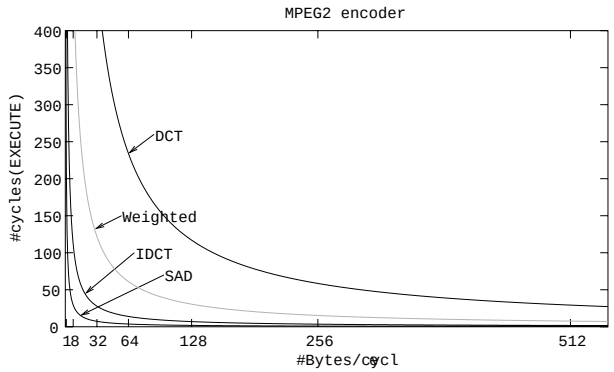

Figure 4. Execution time for different bandwidth

of bytes. Stack operations are not considered as read/write (R/W) operations, as the FPGA implementation most probably will not use a stack. In Table 6, we present the number of read and write memory operations together with the corresponding number of bytes (R_B and W_B). In column R_W_B, the total number of read and written bytes (R_B + W_B) is given.

When we assume automatic transformation and FPGA mapping performed by tools such as Compaan [12], that preserve the memory accesses performed in software, we can analyze the memory bandwidth. As far as the results for DCT are concerned, our calculations are done using the Berkeley implementation of the MPEG2 encoder benchmark including forward DCT-double precision. Figure 4 shows the computed bandwidth requirements for different execution times. Our calculations indicate that DCT is the most demanding function. The reasons of this high bandwidth requirement are: (i) the use of doubles (8 bytes) to minimize information loss during compression, (ii) temporary results are also stored in memory and (iii) the parameters are each time read from memory. If a fast DCT design of around 20-30 cycles is required then around 512 bytes need to be transferred per cycle to fully utilize the DCT unit. Fast SAD and IDCT implementations are less demanding as far as IO is concerned. If SAD is going to be implemented in around 5 cycles, as described in [13], then a bandwidth of 47 bytes per cycle is enough to have a performance gain of 37 % (which is close to the maximal 38 % improvement). When a bandwidth of 128 bytes per cycle is assumed, then a SAD operation can be performed without starvation even in 2 cycles. Similar conclusions can be drawn for IDCT. We finally also computed the bandwidth requirements taking into account the weighted execution times for each function. This curve reflects the requirements of a possible real implementation and suggests that a fast execution time of around 50 cycles for all operations requires a bandwidth of 83 bytes per cycle.

We emphasize that the presented results are based on the assumption that y_{EXEC} is constant (requiring the maximal possible delay) for a specific function, even though it can vary according to the specific input data (e.g. for

Function	Read	R_B	Write	W_B	R_W_B
MPEG2 encoder					
SAD	235	235	0	0	235
DCT	2112	13824	192	1152	14976
IDCT	254	636	128	256	892
VLC I	129	197	1	4	201
VLC II	128	192	0	0	192
MPEG2 decoder					
IDCT	254	636	128	256	892
VLD I	288	962	72	270	1232
VLD II	225	749	55	207	956
JPEG encoder					
VLC	184	547	118	472	1019
DCT	256	1024	128	512	1536
JPEG decoder					
IDCT	344	995	128	320	1315
VLD	1849	6752	539	1926	8678

Table 6. Number of loads/stores performed in the pure software approach

VLD function). Therefore, the actual performance improvements may be higher that presented in this paper.

4 Conclusions

In this paper we presented the design space exploration for some well-known computation-intensive multimedia functions that could be implemented on reconfigurable hardware. The design space exploration allows hardware designers to assess the hardware constraints for the considered functions such as performance boundaries, maximal hardware execution latency, system bottlenecks and IO bandwidth. To this purpose, we have determined for each FPGA implementable operation the maximal cost in order to guarantee a minimal improvement. We also determined the boundaries of these performance improvements and found that for the real implementations described in [1] a 53% reduction of the number of cycles is the expected performance improvement for the MPEG2 encoder. Moreover, our calculations also pointed out that transferring parameters between XRs and GPP registers is not to be a bottleneck. Our calculations suggest that the FPGA-memory data communication can be a bottleneck and access of large blocks (up to 128 bytes) might be a solution to this problem.

Further research will focus on the compiler optimization to hide the FPGA configuration latency and to exploit the parallel operation executions on FPGA.

References

[1] S. Vassiliadis, S. Wong, and S. Cotofana, The MOLEN $\rho\mu$-Coded Processor, *FPL*, Springer-Verlag LNCS, vol. 2147, Belfast, UK, 2001, 275-285.

[2] F. Campi, M. Toma, A. Lodi, A. Cappelli, R. Canegallo, and R. Guerrieri, A VLIW Processor with Reconfigurable Instruction Set for Embedded Applications, *ISSCC Digest of Technical Papers*, 2003, 250-251.

[3] J.M.P. Cardoso and H.C. Neto, Compilation for FPGA-Based Reconfigurable Hardware, *IEEE Design & Test of Computers*, 20(2), 2003, 65-75.

[4] M.B. Gokhale and J.M. Stone, Napa C: Compiling for a Hybrid RISC/FPGA Architecture, *Proc. IEEE Symp. on FCCMs*, Napa, California, 1998, 126-137.

[5] S. Vassiliadis, S. Wong, G.N. Gaydadjiev, K. Bertels, G.K. Kuzmanov, and E. Moscu Panainte, The Molen Polymorphic Processor, *IEEE Transactions on Computers*, Nov. 2004.

[6] S. Vassiliadis, G.N. Gaydadjiev, K. Bertels, and E. Moscu Panainte, The Molen Programming Paradigm, *Proc. of the Third International Workshop on Systems, Architectures, Modeling, and Simulation*, Springer-Verlag LNCS, vol. 3133, Samos, Greece, 2003, 1-7.

[7] E. Moscu Panainte, K. Bertels, and S. Vassiliadis, Compiling for the Molen Programming Paradigm, *FPL 2003*, Springer-Verlag LNCS, vol. 2778, Lisbon, Portugal, 2003, 900-910.

[8] S.K. Rajamani and P. Viswanath, A Quantitative Analysis of the Processor-Programmable Logic Interface, *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, 1996, 226-234.

[9] F. Cavalli, R. Cucchiara, M. Piccardi, and A. Prati, Performance Analysis of MPEG-4 Decoder and Encoder, *Proc. of International Symposium on Video/Image Processing and Multimedia Communications*, Zadar, Croatia, 2002, 227-231.

[10] J. Fritts, W. Wolf, and B. Liu, Understanding Multimedia Application Characteristics for Designing Programmable Media Processors, *SPIE Photonics West, Media Processors* , San Jose, CA, 1999, 2-13.

[11] M. Mercaldi, M.D. Smith, and G. Holloway, The Halt Library, *The Machine-SUIF Documentation Set*, Hardvard University, 2002.

[12] A. Turjan, T. Stefanov, B. Kienhuis, and E. Deprettere, The Compaan Tool Chain: Converting Matlab into Process Networks, *Proceedings of DATE*, Paris, France, 2002, 258-264.

[13] S. Vassiliadis, E.A. Hakkennes, S. Wong, and G.G. Pechanek, The Sum-of-Absolute-Difference Motion Estimation Accelerator, *Proceedings of the 24th Euromicro Conference*, Vasteras, Sweden, 1998, 559-566.