HARTES TOOLCHAIN EARLY EVALUATION: PROFILING, COMPILATION AND HDL GENERATION

Koen Bertels, Georgi Kuzmanov, Elena Moscu Panainte, Georgi Gaydadjiev, Yana Yankova, Vlad Mihai Sima, Kamana Sigdel, Roel Meeuws, Stamatis Vassiliadis

Computer Engineering Laboratory, EEMCS
Delft University of Technology, The Netherlands
http://ce.et.tudelft.nl
Email: {K.L.M.Bertels, G.K.Kuzmanov, E.Moscu-Panainte, G.N.Gaydadjiev, Y.D.Yankova, V.M.Sima, K.Sigdel, R.J.Meeuws, S.Vassiliadis}@tudelft.nl

ABSTRACT

The aim of the hArtes project is to facilitate and automate the rapid design and development of heterogeneous embedded systems, targeting a combination of a general purpose embedded processor, digital signal processing and reconfigurable hardware. In this paper, we evaluate three tools from the hArtes toolchain supporting profiling, compilation, and HDL generation. These tools facilitate the HW/SW partitioning, co-design, co-verification, and co-execution of demanding embedded applications. The described tools are provided by the DelftWorkBench framework[1]. Experimental results on MJPEG and G721 encoder application case studies suggest overall performance improvement of 228% and 36% respectively.

1. INTRODUCTION

The future embedded applications require heterogeneous platforms combining general- and specific-purpose processors with reconfigurable hardware. In addition, coordinated SW and HW development tools are necessary to assist the designers in the iterative and complex development process. The main goal of the hArtes project is to address this problem developing a methodology and a tool-chain supporting the entire embedded system design flow. The hArtes project addresses optimal and rapid design of embedded systems from high-level descriptions, targeting a combination of embedded processors, digital signal processing, and reconfigurable hardware. The project builds a general-purpose toolchain to investigate hardware/software integration for embedded systems. To achieve the main goal of hArtes the following objectives are defined:

- Multiplicity of commercially available languages as toolchain input.
- Partitioning and code transformation algorithms targeting dynamically reconfigurable real time heterogeneous systems.
- "Best fit" mapping onto a heterogeneous reconfigurable system of the input application.
- Development of reconfigurable hybrid platform. The platform, comprising multiplicity of RISCs, DSPs, and reconfigurable logic blocks, employs the HW/SW interfacing mechanisms [1] developed and utilized within the Molen polymorphic processor [2].

In this paper, we evaluate three specific tools from the hArtes toolchain, namely the profiling toolset, the compiler and DWARV - the HDL generation tool. All these tools have been developed within the DelftWorkBench [3] - a semi-automatic tool platform for integrated hardware-software co-design, which provides the required support for the Molen Programming Paradigm [4]. The hardware support of this paradigm is provided by the Molen machine organization [5]. In the hArtes context, the DelftWorkBench tools are further extended to support digital signal processors and application specific hardware, besides the reconfigurable co-processors support. We provide two case study scenaria on G721 and MJPEG applications. Experimental results suggest that application speedups in the order of 1.4 to 3.3 can be obtained easily with the automated design tools being developed in the DelftWorkBench.

This paper is organized as follows. Section 2 provides an overview of the overall hArtes toolchain. Section 3 presents the related work. Section 4 gives the description of the profiling, compilation and HDL generation tools. Section 5 presents the case study of G721 and MJPEG along with the

[1]The DelftWorkBench related research is sponsored by:

- hArtes project (IST-035143) supported by Sixth Framework Programme of the European Community under thematic area "Embedded Systems"
- RCOSY project (DES-6392) supported by STW (Dutch Science Foundation) and ACE (Associated Compiler Experts)

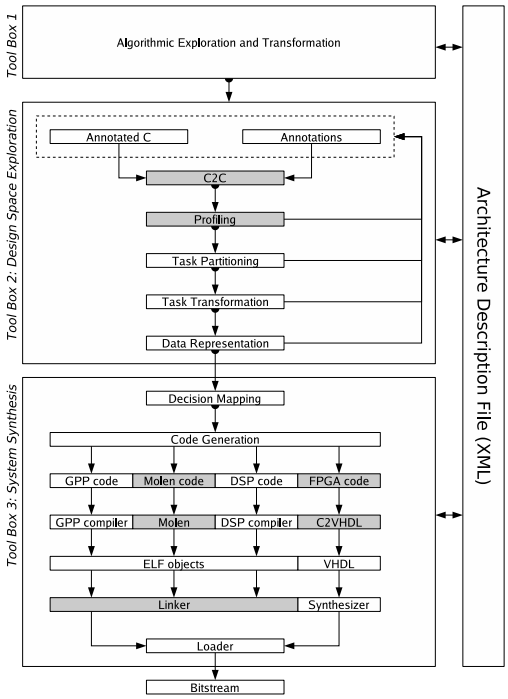

Fig. 1. **hArtes Design Flow**

obtained evaluation results. And section 6 concludes the paper.

2. HARTES OBJECTIVES

hArtes is a three year integrated project intended to develop a methodology and a tool-chain that aims to support the entire design flow for real time, heterogeneous embedded systems having reconfigurable components. The tool chain is general purpose as it takes as input applications written in a variety of commercially available languages and it will produce semi automatically a best of fit mapping of these applications on a heterogeneous reconfigurable platform. The targeted hArtes hardware platform consists of both off the shelf components and proprietary IP kernels. Where the intended workbench is independent of any particular underlying hardware platform, for prototyping purposes we target the Atmel Diopsys, containing an Arm processor and a high performance DSP and state-of-the-art Xilinx Virtex-4 as key components. We are targeting real time systems and applications but in the context of the hArtes project, we focus specifically on immersive audio and car information systems.The hArtes consortium consists of 14 partners equally balanced between industry and academia. The academic partners are mostly involved in developing the toolchain and the industrial partners provide the hardware platform as well as the applications.

An overview of the hArtes toolchain is depicted in Fig 1. The input to the toolchain are multiple high-level application algorithms described in languages such as Matlab or C. The output of the toolchain is semi automatically generated "best fit" mapping of such applications onto a heterogeneous reconfigurable system. The internal representation of the application algorithm is C code annotated with pragmas. The hArtes toolchain is composed of three toolboxes as summarized below:

- The *Algorithm exploration and translation ToolBox* assists designers to instrument and to translate input algorithms described in different formats and languages (e.g., Simulink or graphical entry) into a unified internal description in the C language.
- The *Design space exploration ToolBox (DSE)* provides an optimal hardware/software partitioning of the input algorithm for each reconfigurable heterogeneous system considered. It uses a set of profilers and cost estimators to evaluate various parameters (such as performance, hardware complexity etc) for particular mapping of the kernels onto particular reconfigurable devices.
- The *System synthesis ToolBox (SysSyn)* receives the optimized partitioning of the application algorithm from DSE ToolBox and processes them. The output comprises all generated files, required to map the application algorithm onto the components of the considered reconfigurable heterogeneous system with respect to its partitioning, i.e., program executables, configuration bitstreams, memory images, etc.

In this paper we focus on three specific tools from the hArtes toolchain, namely: profiling, C-code compilation, and HDL generation (shaded blocks on Fig. 1).

3. RELATED WORK

Profiling and Quantitative Model aims to analyze the program statically and/or dynamically to determine relevant information (such as performance, memory bandwidth, power consumption, etc) for design exploration, hardware/software partitioning, and optimization. There are several profiling techniques that can be applied either at compilation time (using static analysis [6] [7])or at run-time (dynamic profiling [8] [9]). Static analysis is less accurate than dynamic profiling as it is based on estimation, while dynamic profiling is slow and requires program intervention. In our approach we combine static and dynamic approaches to develop an efficient profiling and cost estimation tool.

Compilation for reconfigurable architectures can be done in several ways. One common approach is to use standard compilers for general purpose processors (GPP) and impose the programmers to manually modify the assembly

code in order to take into account the reconfigurable hardware. However, this is time consuming and error-prone process which requires deep understanding of both hardware and software features of the target architecture and application. Another common approach (Garp [10], Napa [11]) is to use compiler front-ends with high level optimizations and to generate back C annotated code which is processed by standard C compilers for the target GPP. In consequence, the code quality of the generated code is decreasing and specific low level optimizations cannot be applied. In our approach, we do not require manual interface between the software and the hardware. Moreover, we avoid the library calls overhead and perform low-level specific optimizations.

Automated HDL Generation has been in the research focus for more than 15 years. Our work is similar to the previous efforts in the sense that it combines their advantages, but also differs in several aspects. Unlike Handel-C [12] and some commercial tools like Catapult-C [13] and Impulse-C [14], our tool is oriented towards the software designers. Hence, in-depth hardware design knowledge are not required. We differ from ROCCC [15] and SPARK [16] by targeting a broader application domain and looking into different levels for optimizations. Moreover, we do not impose severe limitations on the accepted C-subset and respect the physical limitations of the available IO bandwidth and access times. Under these limitations, the available operation parallelism is fully exploited. The generated designs have the MOLEN CCU interface, which allows actual execution on a real hardware prototype platform.

The Molen Polymorphic Processor is established on the basis of the tightly coupled co-processor architectural paradigm [5] [17]. Within the Molen concept, a general purpose core processor controls the execution and reconfiguration of reconfigurable co-processors (RP), tuning the latter to various application specific algorithms. An operation, executed by the RP, is divided into two distinct phases: *set* and *execute*. The detailed explanation of these instruction is presented in [5]. Generally speaking, the Molen co-processors are not limited to be only reconfigurable implementations, they can actually be various types of augmenting hardware units. For example, in the context of hArtes, a digital signal processor (DSP) and reconfigurable hardware units are considered as Molen co-processors identically. The Molen machine organization and the Molen programming paradigm are targeted by the DelftWorkBench and hArtes toolchains.

4. TOOLS DESCRIPTION

4.1. Profiling and Quantitative Model

The profiling and quantitative model determines relevant information for design exploration, hw/sw partitioning, and optimization. It identifies core kernels and estimates potential speedup that can be achieved when these kernels are executed on particular reconfigurable hardware. In our approach, we combine both static and dynamic profiling for program analysis. Additionally, we estimate initial cost of hardware mapping and predict upper and lower bounds of speed up possible for the whole application.

Profiling: The main goal of the dynamic code analysis is to determine which parts of an application are computationally intensive when executed on a GPP. The coarse grain profiling inspects the application at function level and gathers information such as CPU time or the execution frequency of these functions. The result of such analysis can be a large list of kernel functions for complex applications. Based on the selection criteria provided (such as performance, area, power consumption, memory etc.), the profiler identifies the core kernels and constructs the candidate function lists for hardware mapping. These candidate functions can be analyzed further at fine-grain level (such as basic block or statement level) to gather detailed information such as number of memory accesses, loop nesting level, etc. The output of the profiler is included in the architecture description file and in the C code, where candidate functions are annotated with the profiling information.

Quantitative Model: The static profiling information is fed into a quantitative prediction model for hardware/software partitioning [18]. This linear model predicts area and delay using software complexity metrics, which are values that represent application structure. An example is the Cyclomatic Complexity [19], which stands for the number of decisions in the code and thus gives some indication of the control intensity of the code.

4.2. Compilation

The role of the compiler is to allow an easy integration while maintaining the best performance for a specific reconfigurable architecture. The Architecture Description File provides the compiler with profile information, area requirement by the kernels and the other specific features of the architecture. Using this information and specific optimization algorithms it generates code for the GPP, extended to accommodate the Molen Programming Paradigm. The compiler is based on GNU C compiler infrastructure 4.1 and it uses a modified version of the PowerPC backend. The compiler identifies the functions that are candidate for hardware execution based on pragmas present in the source file. Using the information available in the Architecture Description File, compiler applied optimizations and scheduling algorithms (see [20] [21]). For the C code in Figure 2 the generated code is presented in Figure 3.

4.3. VHDL Generation

The automated HDL generation within the DelftWorkBench is provided by the DWARV toolset [22]. This toolset accepts

```
...
#pragma call_fpga IntArithDct_In
int DCT_in(int block_i, int c1_i,
                      int c2_i, int c3_i) {
...
}
int DCT (int *block, int *outdata) {
   return DCT_in(block, &(c1[0][0]),
               &(c2[0][0]),&(c3[0][0]));
}
```

Fig. 2. **Application Code**

```
        mtdcr 0x0056,29
        mtdcr 0x0058,11
        mtdcr 0x0059,9
        creqv 6,6,6
        sync
        nop
        nop
        nop
        bl .L42
.L42:
        .long 436207665 # encoding for EXECUTE
        nop
        lwz 0,28(1)
        mfdcr 3,0x0056
```

Fig. 3. **Assembly Code**

as input pragma annotated C code. The annotation specifies the code segments to be implemented in the hardware. Currently, there are syntax restrictions imposed on the accepted input code. Nevertheless, these restrictions do not lead to semantic limitations. The toolset consists of two modules, DFG Builder and VHDL Generator. The DFG Builder currently only transforms the input C-code into intermediate representation (IR), suitable for hardware mapping. The selected intermediate representation is a hierarchical data flow graph. This graph is further processed by the VHDL Generator. Currently, the tool performs only ASAP scheduling on the input graph. The memory bandwidth and access times are provided to the VHDL Generator as additional input. The currently selected computation model is FSM-based. The generated design is with the MOLEN CCU interface [23], which allows actual execution of the generated designs on a real prototyping hardware platform.

5. CASE STUDY

5.1. Evaluation Setup

To evaluate the proposed tool chain, we have selected two applications from the multimedia and DSP domains. The MJPEG application performs image compression and it exhibits the characteristics of the streaming applications, namely regular processing of the input data. The G721 encoder performs audio compression. Although, this application also operates on a streaming input, the performed processing has irregular and control dominated characteristics. To identify the application kernels, dynamic profiling was performed under Montavista Linux 2.4.20 on Xilinx ML 310 board (PowerPC 405). The identified kernel set was further analyzed statically for a final selection of the most profitable code segments to be made. After the application partitioning, the software part was compiled by the MOLEN compiler and the hardware segments were translated by DWARV into VHDL designs. The designs were further synthesized for Xilinx Virtex II Pro (XC2VP30-7FF896) in the Xilinx ISE 8.1 design environment. The hybrid application execution was performed on the MOLEN prototype with PowerPC as GPP, operating on 300MHz. The application kernels implemented as CCUs were executed on 100 MHz.

Table 1. **Resource Estimates for Selected Kernels**

Name	Slices	Flip-Flops	LUTs
fmult	2055	2021	3219
update	8806	6376	14448
DCT	2335	2735	3197
Virtex-II-Pro(XC2Vp30)	13676	27392	27392

5.2. Evaluation Results

Profiling and Quantitative Model: For this experiment we used GNU gprof [9] as a profiling tool. We profiled the benchmark applications with optimization level O3, which resulted with a set of kernels for hardware mapping. For G721 application, we identified two functions $fmult$ and $update$ as candidate kernels and for MJPEG we identified DCT as a candidate kernel. Furthermore, we used our quantitative model to verify if the hardware mapping of these functions satisfies the given constraints on the hardware. The software metric values were determined using our metrication tool and served as input to the quantitative model. The predictions and limitations are shown in Table 1. Compared with the synthesis results in Table 3, the estimates show a considerable error, i.e. 29%-170%. Despite the relatively large error of the model, we can predict which functions fit in the FPGA. Table 1 suggests the candidate kernels identified from profiling can easily fit on the FPGA. Also, $fmult$ and $update$ can fit together on the FPGA, which might reduce the reconfiguration latency of the G721 application.

After the kernel selection, the applications were executed on the target platform without operating system. In addition, hardware timers were used to record the corresponding execution times. In such way, more precise profile of the applications and the selected kernels was derived. The profile data is reported in Table 2. The first column shows the percentage of the application execution time spent in the

Table 2. Profiling of Selected Functions

Name	Time (%)	Cycles	Potential Speedup
fmult	39.67	59684832	1.65
update	38.03	57608874	1.61
DCT	82.15	27819000	5.60

corresponding kernel. The second column shows the total processor cycles for each kernel execution. The last column shows the maximum overall application speedup that can be achieved by speeding up the corresponding kernel. This maximum is the theoretical speedup limit as constituted by the Amdahl's law [24].

Compilation: After hardware/software partitioning, the MOLEN compiler processes the code to be executed on the GPP, placing the necessary configuration and hardware execution instructions. For this evaluation, we used the compiler based on GNU C compiler infrastructure 4.1 which uses modified version of PowerPC backend generation for GPP code generation. The optimization level used for the compilation was O3 which provided the best performance of the application in software.

VHDL Generation and Synthesis: The automated HDL generation was performed by the DWARV toolset, running under Fedora Core 2.6 Linux on AMD Athlon 64 3200+ Processor. The generation time of each kernel is measured using the Linux $time$ utility and reported in seconds. The generated designs were further synthesized for Xilinx Virtex II Pro (XC2VP30) in the Xilinx ISE 8.1 design environment. The generation time and synthesis estimations are reported in Table 3. As it can be observed, although the designs are generated automatically, they are small enough, hence more than one design can reside simultaneously on the device. Another observation that can be made, is that the generated designs can operate on a higher frequency than the one used in the current experiments.

Table 3. HDL Generation Time & Synthesis Estimation

Kernel	HDL.Gen.Time [sec]		Synthesis.Est	
	SUIF2DFG	DFG2VHD	Slices	Freq(Mhz)
fmult	0.495	0.111	1049 (7%)	129.659
update	5.930	0.586	3259 (23%)	164.36
DCT	2.330	0.150	3307 (24%)	100.197
Device Capacity	N/A	N/A	13696(100%)	N/A

Execution: The software part of the applications is compiled with O3 optimization level and is executed on a PowerPC (PPC), which operates at 300MHz. The hardware execution time of the kernels, presented in Table 4, are in PPC cycles for a single run of the kernel. These times are recorded through actual execution on a real hardware platform. As the execution times of $fmult$ and $update$ functions depend on the input data, the reported cycles are the average of the cycles for each kernel execution.

Table 4. Kernels Execution Time

Kernel	SW execution cycles	HW execution cycles	Speedup
fmult	481	378	1.27
update	3714	1218	3.05
DCT	386375	59519	6.49

The modest kernel speedup for $fmult$ is mainly due to the differences in the optimization efforts between the DWARV toolset and the GCC compiler. In its current development stage, DWARV only offers a limited set of optimizations (e.g. scalar replacement etc.). On the other hand, GCC is used with O3 optimization level, which translates to aggressive optimizations applied on the source code. Additionally, the function is static, hence inlining as well as parameter transfer optimizations are possible. The use of static array also allows loop unrolling. Moreover, the encoded functionality is sequential, which offers very small amount of operation level parallelism to be exploited in the hardware. The $update$ function offers higher amount of operation level parallelism, exposed by DWARV if-conversion. The speedup for the DCT kernel is the highest mainly due to the fact that the hardware implementation uses temporary arrays mapped to logic cells, while the software version is forced to use memory mapped arrays as the number of GPP registers is limited. Another reason is the different function call overhead. The static functions allow GCC to optimize the invocation overhead, through inlining and parameters transfer optimizations. Those optimizations, however are not possible to be applied for the hardware version of the functions, which leads to bigger invocation overhead.

Table 5. Overall Application Speedup

Application	Cycles	Speedup	Potential Speedup	Efficiency
SW G721	151344663			
HW fmult	152349759	0.99 (-1%)	1.65	-1.5%
HW update	111673665	1.36 (36%)	1.61	59.01%
SW MJPEG	33859626			
HW DCT	10326096	3.28 (228%)	5.60	49.56%

Although the hardware implementation of these functions was automatically generated without aggressive optimizations, the obtained overall application performance improvement is 36% for G721 and 228% for MJPEG, respectively. These results are presented in Table 5. The last column indicates how close to the theoretical maximum, is the achieved speedup. It is calculated as $(S-1)/(S_p-1)*100\%$, where

S is the measured speedup and S_p is the potential maximum speedup as constituted by the Amdahl's law [24].
Due to slow hardware implementation and different calling convention for the software and hardware execution, the $fmult$ does not offer any performance improvement. Therefore, the $fmult$ kernel is not considered for automatic hardware systhesis. There are two options for this kernel either leave it as software one or manually design a supporting hardware accelerator and include it as a library HDL element in the toolchain. The $update$ kernel offers 36% application improvement, which however constitutes more than 80% of the theoretically achievable speedup. As this kernel comprises 38% of the application, speedup higher than 1.61 cannot be achieved. The DCT kernel offers the higher application improvement due to the larger kernel speedup as well as the higher percentage of the application time spent in the kernel.
The presented case study shows that the Hartes toolchain can be used to achieve significant speedup with minor design efforts. Currently, the programmer is required only to guide the kernel selection in the profiling phase, while the compilation of the whole application and the hardware implementations of the selected kernels are automatically generated by the presented tools.

6. CONCLUSIONS

In this paper, we evaluated the profiling, compilation and HDL generation tools from the DelftWorkBench project. The dynamic profiling unit identified $fmult$ and $update$ as kernel sets for G721 and DCT as a kernel function for MJPEG for HW mapping. With the quantitative model, we predicted that these kernels can fit onto the given FPGA. The compiler compiled the code segments for GPP and it applied a set of optimizations on the application. Finally, HDL generation unit provided VHDL designs of the above mentioned kernels. The obtained results showed that the overall improvement of 36% for the G721 and 228% speedup for MJPEG application.
In our future work, we will introduce new profiling criteria (such as power, memory access) and extend the quantitative model with additional targets(such as delay, interconnect, power). In the compilation phase, we will add new compiler optimizations that will fully exploit the available parallelism in each application, while aggressive optimizations will be introduced in the VHDL generation tool along with design patterns and hardware grammar.

7. REFERENCES

[1] G. Kuzmanov and S. Vassiliadis, "Arbitrating instructions in an $\rho\mu$-coded ccm," in *Proceedings of the 13th International Conference on Field Programmable Logic and Applications (FPL'03)*, September 2003, pp. 81–90.

[2] G. Kuzmanov, G. N. Gaydadjiev, and S. Vassiliadis, "The molen processor prototype," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2004)*, April 2004, pp. 296–299.

[3] Delft workbench. Online: http://ce.et.tudelft.nl/DWB/.

[4] S. Vassiliadis, G. N. Gaydadjiev, K. Bertels, and E. Moscu-Panainte, "The molen programming paradigm," in *Proceedings of the Third International Workshop on Systems, Architectures, Modeling, and Simulation*, July 2003, pp. 1–10.

[5] S. Vassiliadis, S. Wong, G. N. Gaydadjiev, K. Bertels, G. Kuzmanov, and E. MoscuPanainte, "The molen polymorphic processor," *IEEE Transactions on Computers*, pp. 1363–1375, November 2004.

[6] J. Zhu and S. Calman, "Context sensitive symbolic pointer analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circ uits and Systems*, vol. 24, no. 4, pp. 516–531, April 2005.

[7] P. G. Kjeldsberg, F. Catthoor, and E. J. Aas, "Data dependency size estimation for use in memory optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 7, pp. 908–921, 2003.

[8] A. Srivastava and A. Eustace, "Atom: a system for building customized program analysis tools," *SIGPLAN Not.*, vol. 39, no. 4, pp. 528–539, 2004.

[9] S. L. Graham, P. B. Kessler, and M. K. McKusick, "gprof: a call graph execution profiler," in *SIGPLAN Symposium on Compiler Construction*, 1982, pp. 120–126.

[10] T. J. Callahan, J. R. Hauser, and J. Wawrzynek, "The garp architecture and c compiler," *Computer*, vol. 33, 2000.

[11] Gokhale, M. Stone, and J.M., "Napa c: Compiling for a hybrid risc/fpga architecture," in *FCCM '98: Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, 1998, p. 126.

[12] Handle-c language reference. Online: http://www.celoxica.com/.

[13] C-based design. Online: http://www.mentor.com/products/c-based_design/.

[14] Impulse-c. Online: http://www.impulsec.com/.

[15] Riversite optimizing complier for configurable computing. Online: http://www.cs.ucr.edu/ roccc/.

[16] Spark: A parallelizing approach to the high-level synthesis of digital circuits. Online: http://mesl.ucsd.edu/spark/.

[17] S. Vassiliadis, S. Wong, and S. D. Cotofana, "The molen $\rho\mu$-coded processor," in *in 11th International Conference on Field-Programmable Logic and Applications (FPL), Springer-Verlag Lecture Notes in Computer Science (LNCS) Vol. 2147*, August 2001, pp. 275–285.

[18] R. Meeuws, Y. Yankova, K. Bertels, G. Gaydadjiev, and S. Vassiliadis, "A quantitative prediction model for hardware/software partitioning," in *Proceedings of the 17th International Conference on Field Programmable Logic and Applications*, Delft University of Technology, Delft, Netherlands, 2007.

[19] T. J. McCabe, "A complexity measure," in *ICSE '76: Proceedings of the 2nd international conference on Software engineering*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1976, p. 407.

[20] E. Moscu Panainte, K. Bertels, and S. Vassiliadis, "Interprocedural optimization for dynamic hardware configurations," in *Proceedings of SAMOS 05*, July 2005, pp. 2–11.

[21] ——, "Compiler-driven fpga-area allocation for reconfigurable computing," in *Proceedings of Design, Automation and Test in Europe 2006 (DATE 06)*, March 2006, pp. 369–374.

[22] Y. D. Yankova, G. Kuzmanov, K. Bertels, G. N. Gaydadjiev, J. Lu, and S. Vassiliadis, "Dwarv: Delftworkbench automated reconfigurable vhdl generator," in *To appear in Proceedings of the 17th International Conference on Field Programmable Logic and Applications (FPL07)*, August 2007.

[23] G. Kuzmanov, G. N. Gaydadjiev, and S. Vassiliadis, "The molen media processor: Design and evaluation," in *Proceedings of the International Workshop on Application Specific Processors, WASP 2005*, September 2005, pp. 26–33.

[24] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of AFIPS 1967 Spring Joint Computer Conference*, 1967, pp. 483–485.