DWARV: DELFTWORKBENCH AUTOMATED RECONFIGURABLE VHDL GENERATOR

Yana Yankova, Georgi Kuzmanov, Koen Bertels, Georgi Gaydadjiev, Yi Lu, Stamatis Vassiliadis

Computer Engineering Laboratory,
Delft University of Technology
email: Y.D.Yankova@tudelft.nl

ABSTRACT

In this paper, we present the DWARV C-to-VHDL generation toolset. The toolset provides support for broad range of application domains. It exploits the operation parallelism, available in the algorithms. Our designs are generated with a view of actual hardware/software co-execution on a real hardware platform. The carried experiments on the MOLEN polymorphic processor prototype suggest overall application speedups between 1.4x and 6.8x, corresponding to 13% to 94% of the theoretically achievable maximums, constituted by Amdahl's law.

1. INTRODUCTION

The DelftWorkBench[1] aims to provide a semi-automatic platform for hardware/software co-design in the context of Custom Computing Machines [1]. It targets the MOLEN polymorphic machine organization [2], [3] and provides support for the MOLEN programming paradigm [4]. The automated VHDL generation within DelftWorkBench is performed by the DWARV toolset, presented in this paper. The input of the toolset is pragma annotated C code without any syntax extensions. A number of restrictions on the C-language currently apply but will be relaxed in the future. However, those restrictions, do not limit the application domains. Rather algorithms with different characteristics from various application domains are automatically generated. DWARV exploits the available operation parallelism in the algorithms. Contrary to related works, the toolset is designed and developed with a view of actual hardware/software co-execution on a real hardware platform. Hence, the generated designs respect the physically available memory bandwidth and interface specification of the MOLEN polymorphic processor prototype. This allows fast and easy verification and evaluation of the designs on a real hardware prototype platform.

In its current state, with only a limited number of available optimizations, DWARV is capable of providing the following functionality:

- No limitations of the application domains. Algorithms with different characteristics are automatically translated and executed.
- Kernel-wise speedups of 9.7 times over the software execution.
- Substantial overall application speedup of up to 6 times over software execution is observed.
- High performance efficiency. The achieved speedup amounts from 13% to 94% of the theoretically possible maximum speedup, constituted by Amdahl's law [5].
- Actual execution on a real hardware prototype platform.

The experiments in this paper are carried out on the MOLEN polymorphic processor prototype[2]. The corresponding executables for the GPP are generated by the MOLEN compiler [6].

The rest of the paper is organized as follows. Section 2 presents the current structure and functionality of DWARV. Section 3 presents empirical evaluation of the tool set on a real hardware prototype platform. Section 4 discusses some related C-to-VHDL projects. Section 5 outlines further research directions and concludes the paper.

2. THE DWARV GENERATOR

As depicted in Fig 1, the DWARV toolset consists of two modules, the Data Flow Graph (DFG) Builder and the VHDL Generator. The input of the toolset is pragma annotated C code. The annotation specifies the code segments to be implemented in the hardware. The goal of DWARV was to provide straightforward generation of VHDL designs considering actual software / hardware co-execution on the MOLEN polymorphic processor. In its current state, a number of restrictions hold for the code that will be mapped on the hardware. Currently, only one dimensional memory addressing is supported. Structures, unions, and floating point types are not supported yet. The iteration and selection statements are limited to for and if statements, respectively. Additionally, control jumps and function calls are not allowed. Although

DelftWorkBench is sponsored by the hArtes (IST-035143), the MORPHEUS (IST-027342), and RCOSY (DES-6392) projects

[1] http://ce.et.tudelft.nl/DWB

[2] http://ce.et.tudelft.nl/MOLEN/Prototype

syntactically restricted, this C subset does not impose severe limitations on the supported functionality. The unsupported constructs can be substituted preserving the statement semantic (e.g., $while$ loop can be substituted with for loop). In contrast to other hardware compilers, support for algorithms from different application domains is provided. A final requirement is that the DWARV compiler relies on a pragma annotation to identify the functions that need to be translated into VHDL.

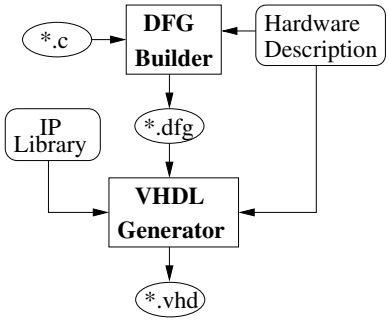

Fig. 1. DWARV Toolset

The input code first is processed by the **DFG Builder**. This tool is currently implemented as a pass within the SUIF2 compiler framework[3]. The purpose of this module is to perform high-level hardware-independent optimizations on the code and to transform it into an intermediate representation (IR), suitable for hardware mapping. Currently, the set of implemented optimizations includes simplified scalar replacement, static single assignment, common sub-expression elimination, and dead code elimination. The output IR is a hierarchical data-flow graph (HDFG). This is a directed acyclic graph with two types of nodes: simple and compound. The simple nodes represent arithmetic and logic operations, registers, and memory transfers. The compound nodes represent the loops in the input code and contain sub-HDFG of the loop body. The edges of the graph represent the data dependencies and the precedence order between the operations. Such representation exposes the available operation parallelism and allows for its exploitation at later stages of the generation. The HDFG is further process by the **VHDL Generator**. This tool is currently implemented as a stand-alone console application. Its purpose is to perform low-level hardware-dependent optimizations and to generate the final VHDL code. Currently, the tool only performs As Soon As Possible (ASAP) scheduling on the input graph. This scheduling, however, aims to fully exploit the available operation parallelism, respecting the given memory bandwidth and access latency. Each operation node is assigned a latency expressed in terms of CCU execution cycles. Currently, finer estimation of the latencies of the different operations is not performed. Rather, all of them are assumed to be executed in one CCU cycle. Nevertheless, operations, that are ultimately translated to wires shifting (shift by constant or bitwise-and by constant) are assume to take zero cycles. The load and store nodes are assigned the corresponding memory access times for read and write operations. These times as well as the available memory bandwidth are provided to the VHDL generator as additional input. The output designs are FSM-based with the MOLEN CCU interface [7]. This allows actual execution of the generated designs on a real hardware prototype platform.

An example of the performed translation process is presented in Fig 2. The input C code (Fig 2a) is transformed into the HDFG, shown in Fig 2b. The shaded area in the figure is a compound $loop$ node with the loop body sub-DFG. The edges denote the data dependencies between the operations. The precedence edges are not shown in the figure. The generated graph is further processed and an FSM-based design is generated (Fig 2c).

[3]http://suif.stanford.edu/suif/suif2/

3. EXPERIMENTAL RESULTS

To evaluate the DWARV toolset, 4 kernels from different application domains are selected and the corresponding VHDL designs are generated by DWARV. The designs are further synthesized for Xilinx Virtex II Pro (XC2VP30) in the Xilinx ISE 8.1 design environment. Next, hybrid software / hardware co-execution of the applications with the generated kernels is carried on the MOLEN polymorphic processor prototype. All reported times are expressed in PowerPC cycles. These times are measured using the PowerPC timer that increments at 300MHz frequency. The kernel execution time is measured when running soley on the GPP and when executed by the FPGA. We refer to them respectively as hardware and software execution. The same hold for the entire application taking into account that only part of it is executed on the FPGA. For each measurement, the PowerPC timer base reagister is initialized to 0 and its value is read at the end of the execution. The reported hardware execution times include the time for evaluating and transferring the function parameters. Hence, the hardware / software interface overhead is included in them.

Application profile: The applications and the considered kernels are listed in Table 1. The G721 application is from the MediaBench benchmark suite [8]. The kernels are selected after profiling the applications. The respective profiles are presented in the third column of Table 1. The first number specifies the percentage of the application execution time spent in the corresponding kernel. The second number indicates the call counts for each kernel. These profiles are derived after execution on the MOLEN prototype. The application execution times and the kernel execution times are measured in separate runs to avoid skewing of the profil-

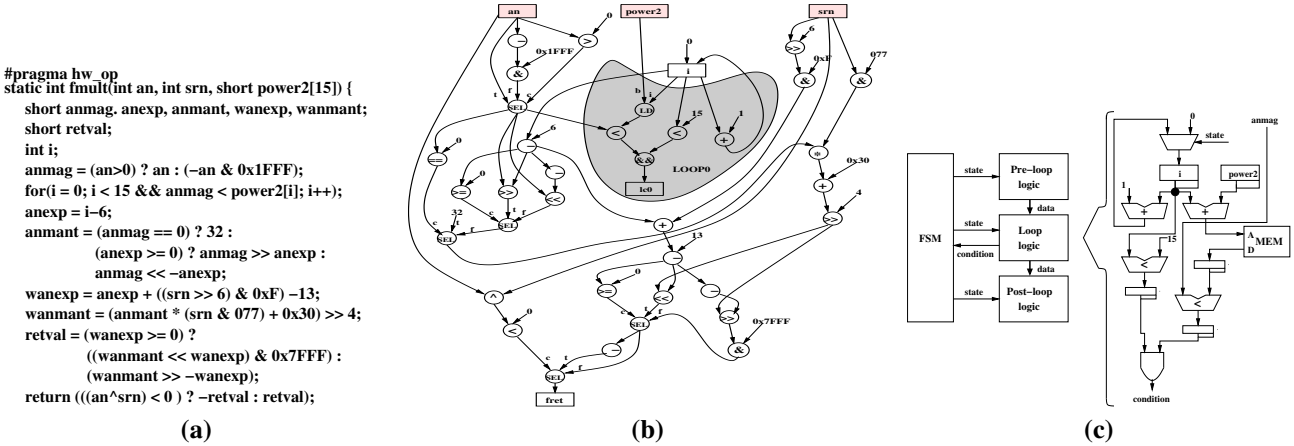

Fig. 2. G721 encoder - $fmult$: (a) C code; (b) HDFG; (d) schematics

Application	Kernel	Profile	Theoretical Speedup Limit
G721 encoder	update	33.40% / 2000	1.5
	fmult	46.28% / 16000	1.86
G721 decoder	update	28.31% / 4000	1.39
	fmult	56.31% / 32000	2.29
MJPEG	DCT	79.05% / 72	4.77
-	AES	97.81% / 192	45.65

Table 1. Applications Profile

Kernel	Software, PPC cycles	Hardware, PPC cycles	Speedup
fmult (e)	23720954	6444656	3.68
fmult (d)	55263398	13442072	4.11
update (e)	17121714	2170212	7.89
update (d)	27784666	3816908	7.28
DCT	37740312	3876408	9.74
AES	2209728	281280	7.86

Table 3. Kernels Execution Time

ing results due to the timer manipulation overhead. The last column of the table shows the maximum overall application speedup that can be achieved according to Amdahl's law [5].

The considered kernels exhibit different characteristics. The implementation of the Rijndael algorithm (AES) requires random memory accesses. The DCT, on the other hand, performs regular processing on the input stream. The $fmult$ function performs table look-up and the $update$ function is control dominated processing of multiple scalar variables.

Kernels synthesis: The VHDL generation and the synthesis of the kernels are performed under Fedora Core 2.6 Linux on AMD Athlon 64 3200+ Processor. The times necessary for DWARV to generate the VHDL for each kernel are reported in the seventh column of Table 2. The first number is the time necessary to built the HDFG for each kernel. The second number is the time necessary to generate the VHDL designs from the HDFG. These times are measured by the Linux `time` utility. The first number in the next column gives the number of C code lines for each kernel. The second number in the same column gives the corresponding lines of VHDL code. From these numbers the declaration lines are excluded. The last column of the table shows the synthesis times (first number) and the times necessary to build the final bitstreams (second number) by the Xilinx tools for each kernel. The synthesis estimations of necessary resources and execution frequency for each kernel are reported in Table 2.

Kernel Performance Improvement: To evaluate the kernel performance improvement, the applications are executed on the MOLEN prototype. The software parts of the applications are executed on PowerPC 405. The CCUs frequency is set to 100MHz. The software and the hardware kernel execution times are reported in Table 3. These times are recorded through actual execution on a real hardware platform. The presented times are the total (sum) cycles spent in the corresponding kernel during the application execution. The execution times of $fmult$ and $update$ functions depend on the input data, which explains the difference between the execution time of the functions within the encoder and decoder applications.

In order to evaluate the quality of the generated designs and to outline future directions for DWARV improvement, a comparison with the manual implementation [9] of the AES kernel is made. The hand-crafted implementation is 8 times smaller than the automated one and operates also at a higher frequency. The achieved kernel speedup is 5.5 times lower than the speedup achieved by the manual design for the same

Kernel	Slices	Flip Flops	LUTs	MUL18X18	Frequency, MHz	DWARV time, sec	Code Lines	Xilinx tools time, sec
update	2779 (20%)	2939 (10%)	4298 (15%)	0	169.926	5.930 / 0.586	127 / 1758	98.332 / 507.117
fmult	1029 (7%)	835 (3%)	1769 (6%)	2 (1%)	129.659	0.495 / 0.111	21 / 445	81.177 / 333.816
DCT	3307 (24%)	3971 (17%)	4965 (18%)	40 (29%)	100.197	2.330 / 0.150	22 / 776	78.074 / 679.791
AES	4181 (30%)	4295 (15%)	6097 (22%)	0	155.130	10.771 / 0.434	74 / 1708	99.945 / 652.226
Device Capacity	13696	27392	27392	136	N/A	N/A	N/A	N/A

Table 2. Synthesis Time and Synthesis Estimations

Application	SW, cycles	HW, cycles	Speedup	Efficiency
G721 E, fmult	51258516	34080767	1.5	58.51%
G721 E, update	51258516	36983874	1.39	76.95%
G721 E, both	51258516	N/A	2.69	43.17%
G721 D, fmult	98137145	57091687	1.72	55.78%
G721 D, update	98137145	71477821	1.37	**94.44%**
G721 D, both	98137145	N/A	3.03	36.95%
MJPEG	47743441	16275541	2.93	51.25%
AES	2259217	334117	6.76	12.90%

Table 4. Overall Application Speedup

input data. This is mainly due to the fact that DWARV currently assumes that all data, including the constant tables, reside in shared memory, where as in the manual implementation the constant data is available on the FPGA. In fact, the particular algorithm implementation uses five constant tables with precomputed values. Four of those tables are accessed in a loop. This translates to multiple memory accesses in the kernel. As the shared memory in the current MOLEN prototype is a single port memory, those accesses prevent parallelization of the algorithm. The manual design on the other hand implements the constant tables in multi-port on-chip memories. This allows four memory accesses to be executed in parallel, which translates to roughly four times faster design in the particular case. This case study shows one possible future optimization, namely privatization of global constant data, to be implemented in DWARV. Such optimization would help in some cases to bring the quality of the automated designs closer to the one of the manual design, if not in terms of area, at least in terms of performance.

Overall application speedup: The lower kernel speedup of the automated designs, however, does not necessary translate to lower speedups application-wise. The achieved overall application speedup is reported in Table 4. To measure this speedup, each application is executed once with the software version and once with the hardware version of the kernel on the MOLEN prototype with the same input data set. The measured times are reported in Table 4. The performance efficiency, reported in the last column of the table, indicates how close the speedup is to the theoretical maximum (the last column of Table 1). The efficiency is computed by the formula: $((Sm-1)/(St-1)) * 100$, where Sm is the measured application speedup and St is the theoretical limit. Such efficiency computation allows possible slowdown of the application also to be captured. The differences in the efficiency of the achieved application speedups is the natural corollary from Amdahl's law, which suggests that when a larger part of the execution time is considered, the higher the potential overall speedup is. The AES kernel comprises $\sim$98% of the application execution time. Therefore, although the highest speedup is achieved for that application, the efficiency is the lowest. On the other hand, the $update$ kernel offers the lowest application speedup but it is the closest to the theoretical maximum. This is due to the fact that only $\sim$30% of the application is implemented in the hardware. The separate hardware implementation of the $fmult$ and $update$ functions does not bring large performance improvement. Nevertheless, as can be observed from Table 2, although automatically generated, the two designs together occupy less than 30% of the available area. This fact allows for their simultaneous mapping on the hardware without dynamic reconfiguration to be neccessary. It can be computed ¡something¿ the Amdahl's law, that a 3-fold application speedup can be achieved in that case.

4. RELATED WORK

The automated HDL generation is in the focus of multiple research projects. The **ROCCC** project [10] aims parallelization of the computation. Also off-chip memory accesses optimization is considered. The target application domain is streaming applications. In the studied applications, only the DCT kernel exhibits such characteristics. In contrast to ROCCC, DWARV does not limit the application domain. **Streams-C** [11] and **SA-C** [12] also have the limited application domain drawback. In addition, they introduce new syntactical constructs, which requires application re-writing. Our toolset does not introduce new syntax, hence no kernel re-writting is necessary. The SPARK project [13] emphasizes on operation scheduling. More specifically, they propose speculative code motion that aims to increase instruction level parallelism in control-intensive kernels. How-

ever, the compiler assumes that all data resides on the chip and all parameters are passed through IO ports. For the DCT kernel this will translate to 2048 IO ports. The XC2VP30 device has 556 IO buffers. DWARV, on the other hand, respects the physical capacity of the target device. It can provide a performance improvement for any pre-defined I/O constraints. In recent years, several commercial tools that generate hardware from HLL input also appeared. **Handel-C** [14] accepts as input C-like language but to use it one needs advanced hardware knowledge. Impulse-C[15] is commercialization of Streams-C and suffers from the same drawbacks. Catapult-C [16] requires extensive designer input in both the applied optimizations and the actual mapping process. DWARV is oriented towards the software designers and does not require in-depth hardware knowledge.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented the DWARV C to VHDL generator tool. Speedups of up to 9.7x kernel-wise and 6.7x times application-wise over the software execution were observed. For some of the applications the achieved speedup is up to 94% of the theoretical maximum. Where related approaches produce highly optimized designs for narrow application domains, DWARV is designed to support a broad variety of application domains (multimedia, DSP, encryption and more). The generated designs respect the physical constraints (memory bandwidth, access times, IO ports) of existing devices. Moreover, the generated designs were actually tested on a real hardware platform. The performed optimizations, although limited at the current stage, provide reasonable overall application speedup by exploiting the available operation parallelism. A current limitation, which will be addressed in the future, is the absence of hardware reuse, which evidently introduces overhead. Furthermore, we want to introduce design patterns and a hardware grammar derived from them to improve the overall quality of the generated designs.

6. REFERENCES

[1] M. Sima, S. Vassiliadis, S. D. Cotofana, J. T. J. van Eijndhoven, and K. A. Vissers, "Field-programmable custom computing machines - a taxonomy," in *Proceedings of the 12th International Conference on Field-Programmable Logic and Applications (FPL 2002). Reconfigurable Computing Is Going Mainstream*, September 2002, pp. 79–88.

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

[3] 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.

[4] S. Vassiliadis, G. Gaydadjiev, K. Bertels, and E. M. Panainte, "The molen programming paradigm," in *Proceedings of the Third International Workshop on Systems, Architectures, Modeling, and Simulation*, ser. Lecture Notes in Computer Science (LNCS), vol. 3133. Samos, Greece: Springer-Verlag, July 2003, pp. 1–19.

[5] 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.

[6] E. M. Panainte, K. Bertels, and S. Vassiliadis, "The molen compiler for reconfigurable processors," *ACM Transactions in Embedded Computing Systems (TECS)*, February 2007.

[7] 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.

[8] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "Mediabench: A tool for evaluating and synthesizing multimedia and communicatons systems," in *International Symposium on Microarchitecture*, 1997, pp. 330–335. [Online]. Available: citeseer.ist.psu.edu/lee97mediabench.html

[9] R. Chaves, G. Kuzmanov, S. Vassiliadis, and L. A. Sousa, "Reconfigurable memory based aes co-processor," in *Proceedings of the 13th Reconfigurable Architectures Workshop (RAW 2006)*, April 2006, p. 192.

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

[11] M. Gokhale, J. M. Stone, J. Arnold, and M. Kalinowski, "Stream-oriented fpga computing in the streams-c high level language." in *Proceedings of the 8th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2000)*, 2000, pp. 49–58.

[12] W. A. Najjar, A. P. W. Böhm, B. A. Draper, J. Hammes, R. Rinker, J. R. Beveridge, M. Chawathe, and C. Ross, "High-level language abstraction for reconfigurable computing." *IEEE Computer*, vol. 36, no. 8, pp. 63–69, 2003.

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

[14] Handel-c language reference. [Online]. Available: http://www.celoxica.com/

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

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