REC-BENCH: A TOOL TO CREATE BENCHMARK FOR RECONFIGURABLE COMPUTERS

Mahmood Fazlali, Ali Zakerolhosseini

Electrical and Computer Engineering Faculty
Shahid Beheshti University
Tehran, Iran
fazlali@cc.sbu.ac.ir, a-zakeri@sbu.ac.ir

ABSTRACT

Over the last decade, significant attempts have been made to employ reconfigurable computers to accelerate the computation intensive parts (tasks) of the multimedia applications. However, shareing the reconfigurable fabric between computing processes is an important issue and therefore, several design time and runtime mechanisms have been proposed to tackle this problem. One of the basic requirements in these approaches is a real application workload with which the performance of the proposed approach can be measured. Vast majority of previous researches have been evaluated using random generated task sets or one real hardware implementation per task. In this paper, to convince this weakness, we present *Reconfigurable Benchmark* (*Rec-Bench*), a tool to create benchmark suite for reconfigurable computers. The result of applying runtime task scheduling algorithms to the benchmarks in *Rec-Bench* and random generated tasks justifies the usefulness of using *Rec-Bench* for the evaluation of runtime resource management algorithm in reconfigurable computers.

1. INTRODUCTION

Reconfigurable computing offers cost-effective solutions for computationally intensive *kernel loops* through hardware reuse. It exploits parallelism inherent in the multimedia applications and accelerates the execution using the reconfigurable fabric. This leads to the need for scheduling and mapping of *modules* corresponding to the kernel loops on the reconfigurable fabrics [1].

Runtime reconfiguration is an important factor in sharing the FPGA logic area among competing applications. It provides the ability to change the configuration not only within a single application, but also between several applications. To do so, a number of runtime management algorithms have been proposed for reconfigurable computers. In this way, researchers in [2] present a task scheduling algorithm for multi-thread reconfigurable systems. In [3] and [4], hybrid design-time/run-time reconfiguration scheduling is presented.

Although the results indicate increase in system speed-up, most of the previous researches employed randomly generated benchmarks. Even in cases of using real benchmark, only one implementation per task is employed. However, employing one real implementation cannot accurately evaluate the runtime management algorithms. This is because of the trade-off exists between the implementation factors such as hardware configuration time and hardware execution time [5]. Therefore, this paper presents *Rec-Bench* to convince this problem.

The main contribution of this paper is presenting *Rec-Bench*. This is a tool to create real benchmarks to evaluate runtime management algorithms aim at reconfigurable computers. The initial goals of *Rec-Bench* are:

1. Focusing on computational kernel loops in multimedia programs, which can be mapped on and accelerated by reconfigurable fabric.
2. Accurately represent the specification of three different hardware implementations for the module.

The rest of the paper is organized as follows. Next section explains the configuration time and execution time of the kernel loops in reconfigurable systems. Section 3 presents *Rec-Bench*. Section 4 includes the evaluation results and section 5 concludes the paper.

2. CONFIGURATION AND EXECUTION TIMES OF THE KERNEL LOOPS

Overall time to compute the kernel loops (in reconfigurable computer) includes the configuration time of the kernel loops (T_C) in addition to the execution time of the kernel loops (T_E). T_C is the required time to configure the module onto reconfigurable unit. T_E is the required time to execute the kernel loops via the configured module. T_C is equal to:

$$T_C = T_F + T_I \quad (1)$$

978-1-4244-6311-4/10/$26.00 ©2010 IEEE

where $T_F = \sum_{\forall v' \in V'} T_f$ is the configuration time of hardware units and, $T_I = \sum_{\forall v' \in V'} T_{i\ (MUX)}$ is the configuration time of interconnection units in the module [5].

Consider a pipelined module, D, corresponding to a kernel loop K, which have a number of time stages. As illustrated in equation (2), the execution time of a kernel loop is T_e, where N is a number of iterations for the kernel loop K, and T_{clk} is the maximum delay of the time stages in D. T_E for n kernel loops in the program will be the aggregate of T_e.

$$T_e = NT_{clk}\ ,\quad T_E = \sum_{i=1}^{n} T_{e_i} = \sum_{i=1}^{n} N_i T_{clk_i}\ . \qquad (2)$$

A trade-off exists between T_C and T_E. This means shortening T_C in synthesizers increase T_E. On the other hand, it is possible that a configured module on reconfigurable fabric is employed several times to execute the kernel loops [2]. This renders the trade-off important. Hence, having variety of implementations for the module, the scheduler can decide which implementation is suitable to be configured on reconfigurable fabric. Therefore, we are to create different implementation for each module, and these implementations should have various T_C and also T_E.

3. REC-BENCH

This section describes steps of producing modules for the input program in *Rec-Bench*. These modules are suitable for a reconfigurable system where the kernel loops are independent therefore; we can safely ignore the contribution of the communication overhead. Besides, the reconfigurable processor has direct access to the cache or, main memory, like Molen processor [6]. This removes the large storage requirements for the data in the FPGA.

Fig.1 illustrates the steps in Rec-Bench to create the modules. At first, the input program is compiled using the GCC compiler, and profiled so as to determine which kernel loops contributed the most to the program execution time. For each such kernel loop, a Data Flow graph (DFG) is generated from the loop body (Register Transfer Level) RTL code (GCC intermediated representation) [9]. Using RTL instead of machine instructions permitted us to extract the kernel loop code after most machine-independent code optimizations, but before register allocation and machine-dependent optimizations.

Moreover, whenever possible, procedure integration (automatic in-lining) is applied. The section of application code corresponding to a DFG can contain control constructions, such as “if-then”, “if-then-else”, and “switch”. The DFGs are generated using a technique based on if-conversion and using condition bit vectors.

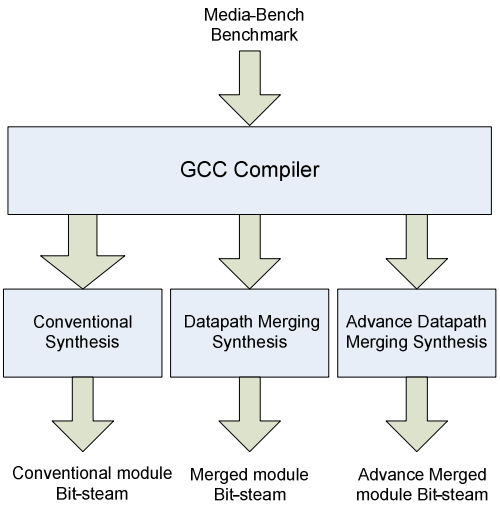

Fig.1. The steps in Rec-Bench to create the benchmarks

We consider up to three kernel loops for each application. The kernel loops/DFGs that require too many resources are discarded, in order to guarantee that the resulting hardware would meet resource constraints of the target FPGA (Virtex5-xc5vlx).

Afterwards for each application, the DFGs are synthesized to create modules in this paper. To this end, we employed three types of synthesizers for the DFGs. The first one is based on the method presented in [10] which uses a conventional synthesis technique to make the module in RTR system. The second one is a technique in [8] that uses maximum weighted clique technique to create the module for the input DFGs. The third one is a technique in [11] that uses the advance merging method to combine the DFGs. The first technique is referred to as Conventional Synthesis. The second technique is referred as Traditional Datapath Merging. The third datapath merging technique is referred as Advance Datapath Merging. After obtaining the bit-stream of the modules, their configuration times were calculated as: configuration time = [(size of bitstream) / (FPGA clock frequency)] [12].

As an example, we employed our tool with the applications in Media-bench [7] as an input program. There are kernel loops in these applications which account for the largest share of the program execution time [8].. These kernels are:

- Jpeg-Encoder and Jpeg-Decoder: Jpeg is a standardized compression method for the images. Jpeg is loosely compression, meaning that the output image is not exactly identical to the input image. Two kernel loops are derived from the Jpeg; Jpeg-Encoder does image compression and Jpeg-Decoder, which does decompression.

Table 1. The configuration time of benchmarks in Rec-Bench.

Benchmarks	Module Configuration time, T_C (*Ms*)		
	Conventional Synthesis	Traditional Datapath Merging	Advance Datapath Merging
MPEG2- Jpeg-Mpeg2-Decoder	11.04	6.39	5.82
Mpeg2-Encoder	4.87	2.66	2.49
Epic-Decoder	5.83	4.11	3.64
Epic-Encoder	7.51	5.68	4.87
G721	10.6	6.39	5.82

Table 2. The execution time of benchmarks in Rec-Bench.

Benchmarks	Execution time of kernel loops, T_E (*Ms*)		
	Conventional Synthesis	Traditional Datapath Merging	Advance Datapath Merging
Mpeg2-Decoder	5.98	8.53	8.56
Mpeg2-Encoder	3.99	4.93	5.22
Epic-Decoder	2.01	2.34	2.43
Epic-Encoder	1.19	1.82	1.94
G721	3.99	4.23	4.64

- Epic-Encoder and Epic-Decoder: The compression algorithms which are based on a bi-orthogonal critically sampled dyadic wavelet decomposition and a combined run-length/Huffman entropy coder. Extremely fast decoding of epic makes it suitable to be employed for portable embedded systems.

- Mpeg2-Encoder and Mpeg2-Decoder: Mpeg2 is the standard for digital video transmission.

- G.721: is for voice compressions.

- Pegwit: A program for public key encryption and authentication. It uses an elliptic curve over GF (2255), SHA1 for hashing, and the symmetric block cipher square.

Table. 1 shows T_C and T_E for Conventional module, Merged module and Advance Merged module in Rec-Bench. As illustrated in this figure, the Advance Merged module has the minimum T_C in all implementation of the benchmark. However, it has the most T_E in comparison to other implementations of the benchmark. On the other hand, the Conventional synthesis has the minimal T_E in each benchmark while, it has the maximal T_C. Having these verities of implementations for the modules let the runtime manager to choose the best candidate based on status of the system.

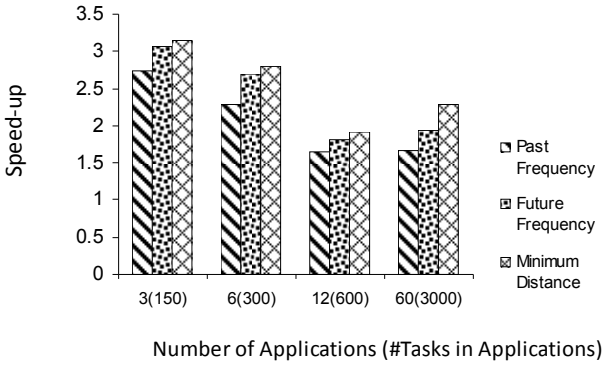

Fig.2. Application Speed-up in random generated benchmarks.

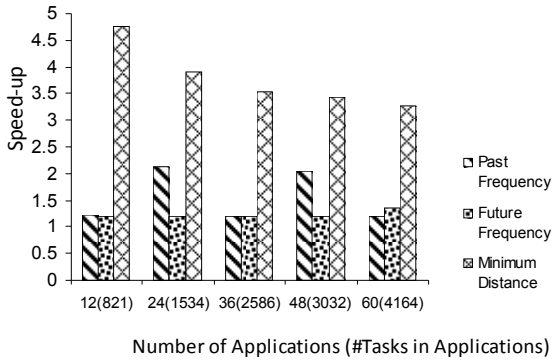

Fig.3. Application Speed-up in Rec-bench.

4. EVALUATION RESULTS

In this section, we evaluate three scheduling algorithms by using hardware generated by *Rec-Bench* as well as randomly generated tasks. The Algorithms are *Past-Frequency*, *distance to the next call* and *Future-Frequency* algorithms. The *Past-Frequency* has been proposed in [2] with the name Most Frequently Used (MFU). *Distance to the next call* and *Future-Frequency* have been presented in [4]. For simulation, we use the discrete event simulator used in [4]. Our target architecture is similar to Molen hardware platform on Xilinx Virtex series.

In case of *Rec-Bench*, we evaluate the system in five different scenarios; 12 applications (821 tasks), 24 applications (1534 tasks), 36 applications (2586 tasks), 48 applications (3032 tasks) and 60 applications (4164 tasks). In case of random generated task, the simulation was run in four scenarios; 3 applications (150 tasks), 6 applications

(300 tasks), 12 applications (600 tasks) and 60 applications (3000 tasks).

As it is shown in Fig.2 and Fig.3, the behavior of the algorithms in *Rec-Bench* is not similar to their behavior in random generated task. This shows that the evaluations using synthetic tasks not always conforms the real world applications behaviors.

5. CONCLUSION

This paper presented *Rec-Bench* to fulfill the need for real application workloads required to evaluate the resource management techniques of the reconfigurable systems. We evaluated the performance of three different scheduling algorithms using the hardware generated by *Rec-Bench.* The results indicate that *Rec-Bench* can be a very good candidate for evaluating runtime management algorithms in reconfigurable computers.

Acknowledgment
This research was supported by the Iran Telecommunication Research Center (ITRC) in the context of the project T/500/3462.

6. REFERENCES

[1] K.H.Hayden, "BORPH: An Operating System for FPGA-Based Reconfigurable Computers," *PhD Thesis,* University of California, Berkeley, July 2007.

[2] W. Fu, K. Compton, "An execution environment for reconfigurable computing," *in Proc. IEEE Symposium on Field Programmable Custom Computing Machines*, pp 149-158, April 2005.

[3] M. Sabeghi and K. L. M. Bertels, "Toward a Run-time System for Reconfigurable Computers: A Virtualization Approach," in *Proc. Design, Automation and Test in Europe Conference* (*DATE09*), pp. 978-081, April 2009.

[4] M. Sabeghi, V. M. Sima and K. L. M. Bertels, "Compiler Assisted Run-time Task Scheduling on a Reconfigurable Computer," in *Proc. 19th Int. Conf. on Field Programmable Logic and Applications* (*FPL09*),pp. 44-50 August 2009.

[5] M. Fazlali, A. Zakerolhosseini, A. Sahhbahrami and G. Gaydadjiev, "High Speed Merged-datapath Design for Run-Time Reconfigurable Systems," in *Proc.* 2009 *Int. Conf. on Field-Programmable Technology* (*FPT09*), pp. 339-342, December 2009.

[6] S. Vassiliadis, S. Wong, G. N. Gaydadjiev, K. L. M. Bertels, Kuzmanov G.K and Panainte E. M, "The Molen Polymorphic Processor," in *Proc. IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1363-1375, November 2004.

[7] C. Lee, M. Potkonjak and W. S Mangione, "Media-bench: a tool for evaluating and synthesizing multimedia and communication systems," in *Proc. thirtieth Annual IEEE/ACM Int. Symposium on Micro-architecture* (*MICRO*), pp. 330–335, December 1997.

[8] M. Fazlali, A. Zakerolhosseini, M. Sabeghi, K. Bertels and G. Gaydadjiev, "Data path Configuration Time Reduction for Run-time Reconfigurable Systems," in *Proc. Int. Conf. on Engineering of Reconfigurable Systems and Algorithms* (*ERSA09*), pp. 323-327, July 2009.

[9] GNU Compiler Collection Internals. [Online]. Available: http://gcc.gnu.org/onlinedocs/

[10] Y. D. Yankova, G.K. Kuzmanov, K. L. M. Bertels, G. N. Gaydadjiev, Y. Lu and S. Vassiliadis, "DWARV. DelftWorkbench Automated Reconfigurable VHDL Generator," in *Proc. 17th Int. Conf. on Field Programmable Logic and Applications* (*FPL07*), pp. 697-701, August 2007.

[11] M. Fazlali, A. Zakerolhosseini and G. Gaydadjiev, "A Modified Merging Approach for Datapath Configuration Time Reduction," in *Proc. 6th International Symposium on Applied Reconfigurable Computing* (*ARC2010*), 17-19 March, 2010.

[12] M. Rollmann and R.Merker, "A Cost Model for Partial Dynamic Reconfiguration," in *Proc. Int. Conf. on Embedded Computer Systems: Architectures, Modeling, and Simulation* (*SAMOS*), pp.182-186, July 2008.