COMPILER-ASSISTED TECHNIQUE FOR RAPID PERFORMANCE ESTIMATION OF FPGA-BASED PROCESSORS

Yan Lin Aung, Siew-Kei Lam, Thambipillai Srikanthan

Centre for High Performance Embedded Systems
Nanyang Technological University
50 Nanyang Drive, Singapore 637553
{layan, assklam, astsrikan}@ntu.edu.sg

ABSTRACT

This paper proposes a compiler-assisted technique to rapidly estimate performance of a wide range of FPGA processors without requiring actual execution on target processor or ISS. Experimental results show that this technique estimates performance of a widely-used FPGA processor with an average error of 2% in the order of seconds.

I. INTRODUCTION

The lack of efficient design methodologies and software tools is recognized as a major impediment to the wide-spread adoption of FPGA technology in the industry. A number of design methodologies and design tools have been proposed and developed so as to exploit the computational capabilities and flexibility offered by modern platform FPGAs. It has now become apparent that electronic system-level approach is necessary to bridge the widening productivity gap between hardware design, software development and semiconductor technology. This has led to the development of high-level synthesis (HLS) tools that allow automatic compilation of algorithm description in high-level languages such as C, C++ and SystemC to register transfer level hardware. HLS approaches vary widely from hardware only implementations of high-level applications to processor-accelerator systems to heterogeneous multi-core systems. Some HLS tools have been developed for domain specific applications such as GAUT [1], ROCCC [2]. Quality of results and usability are two key criteria to measure capability and effectiveness of HLS tools. Recent BDTI evaluation results on two commercial HLS tools: AutoPilot from AutoESL and Synphony C from Synopsys show that current state-of-the-art HLS tools are capable of achieving both criteria. Such advancement plays a pivotal role in design space exploration for FPGA-based embedded systems.

Design space exploration refers to evaluation of possible candidate design instances with varying design trade-offs on performance, hardware area, power, etc. to determine an optimal solution. Profiling software application on target processor and using cycle accurate instruction set simulator are two typical techniques for performance evaluation during design space exploration process to identify suitable configurations for processor customization.

Profiling tools can be categorized into intrusive and non-intrusive methods. Intrusive profiling tools such as gprof [3] insert instrumentation code into the application executable and generate a statistics file upon execution. The insertion of the instrumentation code may lead to inaccuracies in performance evaluation. Non-intrusive tools rely on dedicated hardware peripherals such as performance counters. Both approaches require the application to run on target processor in order to enable the profiler to gather statistics of the application during program execution. Although application profiling could be carried out conveniently for standard desktop computers and workstations, the majority of profiling tools for embedded systems require a comprehensive software and hardware platform setup.

Alternatively, instruction set simulators (ISS) has been used for performance evaluation. The ISS approach as well is time-consuming and imposes several limitations considering the advent of a wide variety of open-source and proprietary configurable FPGA processors. ISS may not be readily available for a selected target processor.

The limitations of existing performance evaluation techniques have led us to develop a rapid and reliable processor performance estimation tool capable of facilitating processor customization for a wide range of FPGA processors. The proposed technique achieves rapid performance estimation in two steps. The application is first profiled with suitable input dataset on the standard desktop computer followed by cross-compilation of the application to generate target processor specific assembly code. In the second step, the profiling output, assembly code and instruction set architecture (ISA) description of the target processor are employed to estimate performance of the application. We choose the LLVM compiler infrastructure [4] due to its comprehensive features for profiling and backend code generation for a wide range of processors. Our estimation framework provides performance estimate of the application in terms of number of clock cycles for the selected target processor within seconds.

The rest of this paper is organized as follows: Section II presents related work on performance evaluation using profiling tools and other performance estimation techniques. Section III introduces our proposed performance estimation framework followed by experimental results and discussions in Section IV. Finally, we conclude the paper in Section V.

II. RELATED WORK

Performance evaluation is indispensible in the design space exploration process. Profiling tools and cycle accurate instruction set simulators are often employed in conventional approaches. Many application profiling tools have been proposed to facilitate design space exploration for applications targeting FPGA-based platforms. Most of them are hardware-assisted and support non-intrusive profiling capability. Profilers in LegUp framework [5], SnoopP [6] and Airwolf [7] profile the C application at the function level whereas FLAT profiler [8] detects most frequently executed loops of the program. Tong et. al. provided quantitative comparisons of profiling tools for FPGA-based embedded systems in [7]. FPGA vendors provide standard gprof tool for their proprietary processors. Altera has a set of comprehensive profiling utilities such as timestamp interval timer and performance counter peripherals. These profiling tools require the software application to run on the target hardware platform to obtain the profiling results. This necessitates a comprehensive setup and such profiling approach is possible only if the target hardware platform is available.

Performance estimation is a promising solution to overcome the limitations imposed by the profiling tools. Early research efforts on performance estimation using POLIS hardware-software co-design system could be found at [9, 10]. Recently, a high-level approach to modeling and verification of digital systems known as transaction level modeling (TLM) has been presented [11, 12]. Performance estimation is achieved in the TLM approach at a high abstraction level and this could lead to high inaccuracies in the estimation.

On the other hand, TotalProf profiler proposed by Gao et. al. bears the most similarity to our framework [13]. The TotalProf employed a virtual compiler backend to resemble the course of target compilation and many of standard yet intricate compiler passes such as code selector, pre-register-allocation scheduler, etc are required by the virtual backend though they are well implemented and supported in LLVM. Instead of having such a virtual backend compiler, we rely mainly on the LLVM and its backend code generator in our proposed technique while achieving high estimation accuracy. Moreover, the performance estimation technique suggested in this paper overcomes the limitations of profiling tools by avoiding the need for ISS as well as the availability of target processor and associated hardware-software setup in order to carry out performance evaluation. The proposed technique can reliably estimate performance of the application within seconds enabling efficient design space exploration.

III. PROPOSED TECHNIQUE

The proposed performance estimation framework is shown in Fig. 1. The framework relies on the LLVM compiler for application profiling and generating the target processor assembly code. Performance estimation is then performed using the execution profile of the application, target processor assembly code and ISA description of target processor.

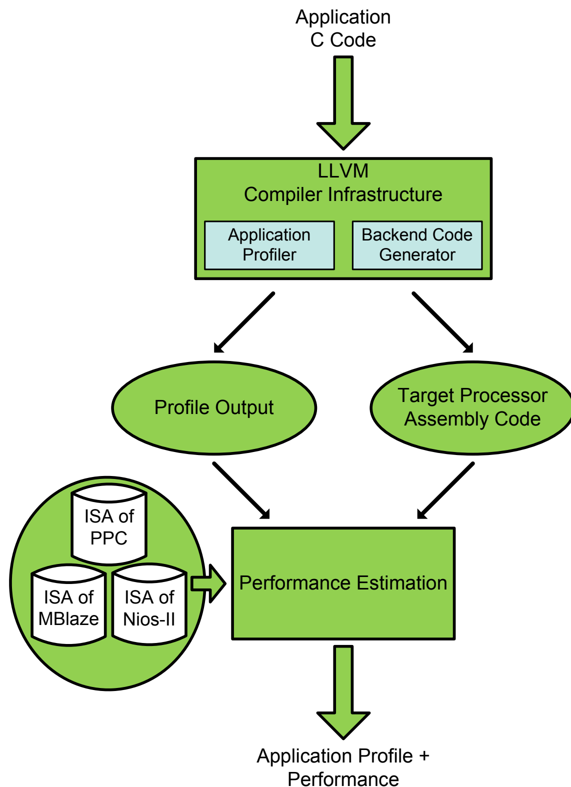

Figure 1: Performance Estimation Framework

A. Application Profiling

The application C code is first compiled with the LLVM compiler into a bitcode file, which is a bitstream container format to store the encoded LLVM intermediate representation (IR). The bitcode file is then executed and profiled using just-in-time compiler and profiling facility of LLVM respectively. The llvm-prof tool is used to extract the execution frequency of basic blocks as well as edges from those basic blocks.

Consider a 32-tap finite impulse response (FIR) filter on 128 data values. The C code for FIR filter from [14] is given in Fig. 2. The output, y[i1], is a weighted sum of the current input sample, x[i1+i2], and 31 previous input samples. The basic block and edge profiling output of the FIR C code from the LLVM profiler is depicted in Fig. 3.

```
/* #define N1 128 */
/* #define N2 32  */
for(i1=0; i1<=N1-N2; i1++){
  for(i2=0; i2<N2; i2++)
    y[i1] = y[i1]+w[i2]*x[i1+i2];
}
```

Figure 2: C Code for FIR Filter

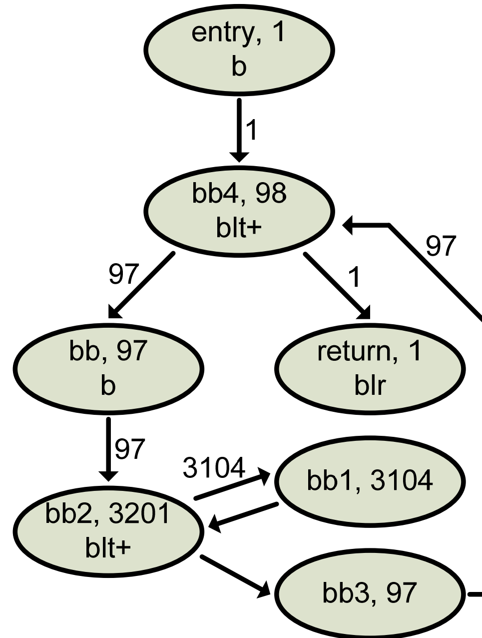

Figure 3: Block and Edge Profiling Output of FIR Filter

The entries in the first line of each node refer to the name and execution frequency of the basic block, whereas the second line provides the branch information of the basic block. The number beside the arrow refers to the edge frequency.

B. Processor Specific Assembly Code Generation

The LLVM compiler provides a backend code generator, llc, which can convert the LLVM IR to either assembly or binary code for a specified target processor. The latest LLVM 2.9 release supports 16 types of target processors, for example, x86, PowerPC, ARM, MIPS, SPARC, etc. Since the release 2.7, the LLVM distribution supports experimental Xilinx's MicroBlaze processor backend code generator.

C. Target Processor ISA

The ISA definition file of target processor consists of a list of instruction mnemonics specific to the target processor with associated instruction class and execution cycles. The LLVM infrastructure uses its own TableGen description to capture details of target architecture. Similar to the machine description file in GCC compiler, LLVM utilizes the TableGen description file to convey information on processor with instruction itineraries. Each itinerary describes instruction

class, stages and operand cycles. The instruction stage represents a non-pipelined step in the execution of an instruction and is composed with length of the stage in terms of machine cycles and available choice of functional units. The LLVM relies on this description file during the instruction scheduling pass to construct the scheduled directed acrylic graph. The ISA description file for our proposed estimation technique is derived from the TableGen description file to represent the target processor architecture.

D. Performance Estimation

The estimation framework extracts the assembly code of each basic block by matching the names of basic blocks in the profiling output and assembly files. For the example in Fig. 3, the estimation tool looks for the corresponding basic block names (i.e. bb1, bb2, etc.) in the generated assembly file and the corresponding assembly codes are extracted.

The total number of clock cycles required for each basic block is computed by summing the number of clock cycles required for all the instructions in each basic block as in (1) where c_k denotes number of clock cycles required for instruction k.

$$bb = \sum_{k=1}^{n} c_k \qquad (1)$$

Performance of C functions in the application can then be estimated by multiplying the number of clock cycles required of each basic block with the respective execution frequency of that basic block, for all basic blocks in the function as in (2). bb_k and $f(bb_k)$ refer to the number of clock cycles and execution frequency of the basic block k respectively.

$$func = \sum_{k=1}^{i} bb_k \times f(bb_k) \qquad (2)$$

In addition, the performance estimation framework must also take into account the conditional branches and prediction penalties as the number of clock cycles required for unconditional branches, conditional branches and branch prediction penalties differs depending upon the branch characteristics. The proposed framework estimates conditional branches by means of utilizing basic block edge frequency information from the LLVM profiling output and static branch prediction flag encoded into the assembly. Branch predictions for our FIR filter example are shown in Fig. 3. There are two conditional branches in the application in which the '+' sign indicates predicted-taken branch whereas the '−' sign indicates predicted not-taken branch. When the application includes calls to standard C library functions, performance estimates of those library functions for the target processor are also required [15]. By taking into consideration the conditional branches, prediction penalties and standard C library function calls, the basic block performance estimation equation in (1) is modified as shown in (3). This has been incorporated in our proposed framework to estimate the performance of the target FPGA-based processor. $c(a_j)$ and $c(l_j)$ refer to number of clock cycles required for nested application function a_j and standard C library function l_j. br denotes the number of additional clock cycles required for branch instructions.

$$bb = \sum_{k=1}^{n} c_k + \sum_{j=1}^{p} c(a_j) + \sum_{j=1}^{q} c(l_j) + br \quad (3)$$

IV. RESULTS AND DISCUSSIONS

A. Experimental Results

Our experiments are based on the Xilinx Virtex-4 FX60 FPGA. The reference system on FPGA is a PowerPC 405 processor with 64KB of data memory, 128KB of instruction memory and UART peripheral. The benchmark applications considered in our experiments are: adpcm, aes, blowfish, FIR, gsm, mips, motion and sha. All but the FIR application are from the CHStone benchmark suite [16]. The LLVM generated assembly code of the application for target processor is compiled to produce the executable, which is initialized to instruction and data memories. We have written an assembly code routine that accesses the time-base registers available on the PowerPC processor for execution time measurement. This measured execution time is used for comparison with the performance estimate from the framework.

Our proposed framework relies on (2) and (3) to estimate performance of the benchmark applications. The PowerPC 405 processor

executes majority of instructions in one clock cycle and unconditional branch instruction takes three clock cycles. The number of clock cycle required for conditional branches are obtained by system simulation with ModelSim and [17]. Performance estimation results in terms number of clock cycles for the benchmark applications, performance measurements from the reference system and absolute estimation error are shown in Fig. 4. Detailed measurement and estimation results are provided in Table 1. The proposed technique is able to estimate performance of all benchmark applications with an average estimation error of 2% and maximum estimation error of 11%.

B. Discussions

In this section, we would like to discuss several issues that will be addressed in our future work. As mentioned in Section III, the availability of LLVM backend for a specific target processor is crucial for our approach. We used the generic 32-bit PowerPC backend, ppc32, for our experiments as there is no exact LLVM backend for PowerPC 405 processor. If there is no LLVM backend available for a specific target processor, there are two possible approaches to carry out processor performance estimation: generate LLVM backend for the target processor and estimating processor performance at the LLVM virtual instruction level. Due to the modular and reusable nature of LLVM compiler infrastructure, the former approach is manageable and less time-consuming compared to writing a backend for other compiler such as GCC compiler. On the other hand, since the LLVM compiler is capable of describing the application using a RISC-like virtual instruction set [4], performance estimation could also be carried out at the virtual instruction level. We are interested to investigate this in future and conduct comparisons on the estimation accuracy with our proposed technique.

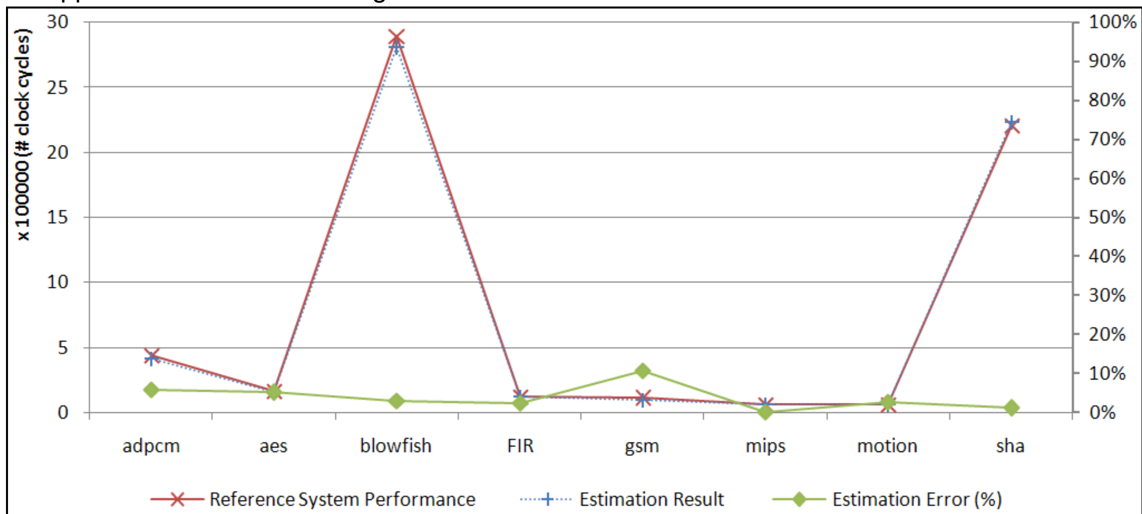

Figure 4. Reference System Performance vs. Estimation Results

Table 1. Detailed Reference System Performance and Estimation Results

Benchmarks	Reference Hardware (# clock cycles)	Performance Estimation Results
adpcm	417890	441972 (106%)
aes	158802	166947 (105%)
blowfish	2806789	2892417 (103%)
FIR	123505	120592 (98%)
gsm	100407	111254 (111%)
mips	61005	60884 (100%)
motion	64638	62966 (97%)
sha	2234678	2209550 (99%)
Average	**100%**	**102%**

In addition, we plan to consider cache events in our framework. The benchmark applications in our experiments execute from on-chip data and instruction memories thus eliminating the effect of cache hits and misses on the performance measurement from our reference system.

The proposed technique is also suited for rapid design exploration to identify optimal configurations for soft-core processors as it can take into account performance improvements or penalties resulting from the inclusion or exclusion of configurable units by using the LLVM's llc static compiler to disable or enable the selection of specific features of the target processor.

V. CONCLUSION

In this paper, we presented a framework for rapid processor performance estimation, which relies on the open-source LLVM compiler infrastructure. The proposed technique overcomes the limitations of existing methods as it does not require the execution of the application on the target processor or ISS. Using the profiling output, ISA description and assembly code of target processor, the framework provides performance estimate of the application in terms of number of clock cycles for the selected target processor. In addition, the proposed method takes into account standard C library function calls and other runtime characteristics e.g. conditional branches and prediction penalties. When compared to the measured performance on the actual target processor, the average estimation error is less than 2% for the applications considered and the estimation can be achieved within seconds. Our proposed technique can also facilitate efficient space exploration to enable application-aware customization of FPGA-based configurable processors.

REFERENCES

1. P. Coussy and A. Morawiec, "GAUT: A high-level synthesis tool for dsp applications," in High-level synthesis : From algorithm to digital circuit: Springer, 2008, pp. 147-170.
2. J. Villarreal, A. Park, W. Najjar, and R. Halstead, "Designing modular hardware accelerators in c with ROCCC 2.0," in Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual International Symposium on, 2010, pp. 127-134.
3. S. L. Graham, P. B. Kessler, and M. K. McKusick, "gprof: A call graph execution profiler," SIGPLAN Not., vol. 39, pp. 49-57, 2004.
4. C. Lattner and V. Adve, "LLVM: A compilation framework for lifelong program analysis & transformation," in Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization, Palo Alto, California, 2004, p. 75.
5. A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, J. H. Anderson, S. Brown, and T. Czajkowski, "LegUp: High-level synthesis for fpga-based processor/accelerator systems," in Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays, Monterey, CA, USA, 2011, pp. 33-36.
6. L. Shannon and P. Chow, "Using reconfigurability to achieve real-time profiling for hardware/software codesign," in Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays, Monterey, California, USA, 2004.
7. J. G. Tong and M. A. S. Khalid, "Profiling tools for FPGA-based embedded systems: Survey and quantitative comparison," JCP, vol. 3, pp. 1-14, 2008.
8. A. Gordon-Ross and F. Vahid, "Frequent loop detection using efficient non-intrusive on-chip hardware," in Proceedings of the 2003 international conference on Compilers, architecture and synthesis for embedded systems, San Jose, California, USA, 2003.
9. K. Suzuki and A. Sangiovanni-Vincentelli, "Efficient software performance estimation methods for hardware/software codesign," in Proceedings of the 33rd annual Design Automation Conference, Las Vegas, Nevada, United States, 1996, pp. 605-610.
10. M. Lajolo, M. Lazarescu, and A. Sangiovanni-Vincentelli, "A compilation-based software estimation scheme for hardware/software co-simulation," in Proceedings of the seventh international workshop on Hardware/software codesign, Rome, Italy, 1999, pp. 85-89.
11. T. Kempf, K. Karuri, S. Wallentowitz, G. Ascheid, R. Leupers, and H. Meyr, "A sw performance estimation framework for early system-level-design using fine-grained instrumentation," in Proceedings of the conference on Design, automation and test in Europe: Proceedings, Munich, Germany, 2006, pp. 468-473.
12. Y. Hwang, S. Abdi, and D. Gajski, "Cycle-approximate retargetable performance estimation at the transaction level," in Proceedings of the conference on Design, automation and test in Europe, Munich, Germany, 2008, pp. 3-8.
13. L. Gao, J. Huang, J. Ceng, R. Leupers, G. Ascheid, and H. Meyr, "TotalProf: A fast and accurate retargetable source code profiler," in Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis, Grenoble, France, 2009.
14. L. Chakrapani, J. Gyllenhaal, W.-m. Hwu, S. Mahlke, K. Palem, and R. Rabbah, "Trimaran: An infrastructure for research in instruction-level parallelism," in Languages and compilers for high performance computing, vol. 3602, R. Eigenmann, Z. Li, and S. Midkiff, Eds.: Springer Berlin / Heidelberg, 2005, pp. 922-922.
15. A. Ray, J. Wu, and S. Thambipillai, "Estimating processor performance of library function," in Embedded Software and Systems, 2005. Second International Conference on, 2005, p. 6 pp.
16. Y. Hara, H. Tomiyama, S. Honda, and H. Takada, "Proposal and quantitative analysis of the CHStone benchmark program suite for practical c-based high-level synthesis," Journal of Information Processing, vol. 17, pp. 242-254, 2009.
17. Xilinx, "PowerPC 405 processor block reference guide," 2010.