From Software to Silicon in Big Data Era

How far are we from Ideal to Practical High Level Synthesis using FPGA?

Chao Wang[1,2], Xi Li[1] , Qi Guo[1], Yuan Xie[2] and Xuehai Zhou[1]
[1] School of Computer Science, University of Science and Technology of China
[2] Electrical and Computer Engineering, University of California at Santa Barbara
[1] cswang,llxx,gqustc,xhzhou@ustc.edu.cn [2]yuanxie@ece.ucsb.edu

***Abstract*—In big data era, the heterogeneous accelerators using Field Programmable Gate Arrays, (FPGAs) has renewed many research interests to improve the speedup and efficiency. However, the gap between high level programming and hardware implementation is bringing significant challenges to the software programmers, especially for those without sufficient hardware skills.**

With the help of the High Level Synthesis (HLS) tools, it is now feasible to generate hardware (register transfer level, RTL) codes from high level software. In particular, HLS tools for FPGAs have made considerable progress over the past few years, and are now sufficiently sophisticated that a developer could create functionally correct implementation with limited understanding of the target hardware. In this case study, we propose a practical evaluation to measure the efficiency of the cutting-edge industrial Vivado tool provided by Xilinx. The RTL efficiency, middleware overheads and running time of the RTL generation are measured respectively.

Experimental results reflect some limitations of state-of-the-art industry tools: 1) the efficiency of the generated code is unsatisfying for non-intuitive applications; 2) When an operating system is incorporated, the scheduling overheads is significant enough (>100us) to hide the speedup for some intuitive benchmark applications; 3) The running time to generate RTL code may take hours or even days.

***Index Terms*—Big Data, High Level Synthesis, FPGA**

I. INTRODUCTION

With the wide application of social computing, mobile computing and networking, it has been a common knowledge that social and enterprise big data has pervaded into our daily lives. Vast volume of data generated at an explosion-like speed and the fast growth rate of the global data are unprecedented [1]. It makes our life more convenient while at the same time also poses significant challenges to computer researchers and scientists. Large data processing and analysis capabilities are far less than the ideal level, which means it needs the capacity of high-speed information transmission, real time processing and data analysis. Furthermore, low value density feature makes data mining more challenging. The data collection and the procession of data mining are quite timing consuming, it will dramatically drag down the performance using conventional computing systems. As a consequence, how to efficiently accelerate the big data application by using heterogeneous accelerators like GPU, FPGA or ASIC is becoming increasingly important, especially in machine learning [2] and neural networks [3].

However, one of the major problems we have to face is that the gap between high-level programmer and the hardware architectures. During the last few years, High Level Synthesis (HLS) has made great progressive achievements forward. By abstracting onto commonly used programming languages such as C, C++ and Haskell, the pool of potential hardware developers has grown by several orders of magnitude. It is essential and necessary to construct an efficient operating system or middleware support to bridging the gap by enabling various coarse grained heterogeneous accelerators [4].

In this paper, we describe our investigation as to whether several leading HLS tools are ready for adoption in an ideal industrial utilization. We have found that while these tools generally are ready, there are still negative (imperfect) results, when they actually are playing a significant role in determining the implementation for application specific domains.

In order to evaluate the industrial HLS and middleware support for FPGA based platforms, one of the major contributions of this work is to propose an FPGA based design flow and framework with profiling and data analysis for big data applications. The framework is able to facilitate researchers to construct a fast prototyping evaluation platform, which integrates the hardware accelerators, peripherals, tool chains, and an operating system support. The platform is targeted at heterogeneous architectures with the following characteristics:

1) Automatic parallelization: Most data-intensive computational tasks are expressed as a running threads to be processed in parallel. Data chosen for the parallel tasks depends on data previously processed and arrived, and the execution starts upon getting the required input data. In these problems, the inter-task dependencies should be analyzed to make tasks run out-of-order.

2) Dataflow execution: Instead of streaming and processing the entire dataset, the computation is composed of multiple dataflow execution phases, each of which continuously examines subsets of data in parallel. The messages exchanged between different phases are regarded as tokens. Whenever the required tokens are ready, the task could be fired, therefore the entire tasks could be executed out-of-order.

3) Component based programming model: The heterogeneous accelerators are abstracted as components, which could facilitate researchers to incorporate heterogeneous functional modules, as well as to ease the burden of the programmers.

4) Based on the hardware platform, we evaluated following features of the state-of-the-art HLS tools: the gap between the efficiency of the RTL generated code and optimized RTL code; the scheduling overheads when porting a Linux operating system to handle the threads; and finally the running time of RTL generation using Xilinx HLS software tools.

The remainder of this paper is organized as below. In Section II we summary the related work. Thereafter Section III details the proposed acceleration architecture, which includes the architecture framework, the programming model, and the thread management in the operating system middleware support. Then in Section IV we show the hardware prototype using Xilinx Zynq FPGA using several test case studies. Experimental results illustrate the imperfect efficiency from the state-of-the-art HLS tools and the considerable scheduling overheads middleware support. Finally, the paper is concluded in Section V.

II. Related Work

A. Architecture Design in Big Data Applications

With the increasing demand of state-of-the-art applications in big data era, the instruction-level parallelism on uniprocessor based architectures would definitely run out-of-performance in the foreseeable future, therefore more researchers are seeking parallelism is shifting from instruction level to task and data level.

Of the coarse-grained parallel architectures, it can be acknowledged that the high-performance, low-power FPGA based computing mode has renewed many research interests during the past few years. For example, Microsoft [20] has successfully adopted FPGA into the datacentre acceleration, which is able to achieve significant speedup for BING search engines. Furthermore, some researchers have also proposed scalable FPGA cluster platforms, such as CUBE [5], Axel [6], FPMR [7], ZCluster [8], SODA [21], and SNNAP[3].

B. Operating Systems Framework

Operating system plays a key role in data management and middleware support. So far there are quite some creditable literatures on parallel multicore architectures, such as Corey [9], fos [10] and Barrelfish [11]. To support heterogeneous reconfigurable accelerators, Hthread [12] presents a multi-threading model for FPGA based reconfigurable systems, which could leverage the heterogeneity between computing resources. Berkeley's BORPH [13] stores the reconfigurable logic configuration information in the executable file, and utilizes FIFO buffer to achieve inter-thread communication. Also, ReconOS [14] is POSIX standard multi-threading operating system on a reconfigurable platform.

C. Programming Model and Runtime

The parallel programming model based on FPGA and GPU based heterogeneous accelerators have been deeply conducted for decades. There are already a good number of creditable heterogeneous parallel programming models such as OpenCL [18], and OpenACC [15], as well as domain-specific languages like OptiML [17] and Delite [16]. These frameworks intend to provide a unified programming interface for heterogeneous multi-accelerator platform, but programmers still need to understand detailed knowledge about the underlying accelerators, and thereby increase the challenges to programming and compiler design complexity.

Although plenty of researches have been devoted to task scheduling to manipulate heterogeneous accelerators, there have been no well-established support conducted for FPGA based accelerators for future data-intensive applications. In this paper we make a brief summary of the stage-of-the-arts

TABLE I. Summary for State-of-the-art on Related Research Fields

Research	Categories	Typical	Strengths	Weaknesses
System Architecture	Heterogeneous FPGA+Accelerators	SNNAP[3], CUBE [5], ZCluster [8], Microsoft [20]	Low power and High efficiency	Low Productivity
Operating System	Symmetric Architectures/ Heterogeneous based on FPGA+Accelerators	Corey [9], fos [10], barrelfish [11], Hthread [12], BORPH [13], ReconOS [14]	Manage symmetric resources efficiently	Do not utilize accelerators efficiently
Programming Model/Runtime Support	Heterogeneous Architectures (CPU+GPU)	OpenACC [15], OptiML [17], Delite[16]	Achieve data level parallelism	Need annotations or new language
	Heterogeneous (FPGA+Accelerators)	OpenCL [18], TaskSuperscalar[19],	Could manage accelerators	Programming Wall

in Table 1. Furthermore, we put concentrations on the FPGA based accelerator architecture, manage the parallel threads running on heterogeneous accelerators, and analyze the dataflow in big data applications.

III. Acceleration Architecture

A. Architectural Framework

The proposed hardware architecture is constructed in a hardware platform that can provide heterogeneous reconfigurable multicore resources such as FPGA. It consists of an application layer and a middleware layer. For hardware implementation several embedded processors are designed as software computing nodes, while various types of hardware intellectual property (IP) cores as hardware nodes, interconnect modules, processor local bus, memory, and peripheral modules. The application layer provides the basic run-time environment and application programming interfaces (API) to tasks. The middleware layer is in charge of system virtualization, task partitioning, mapping, distributing and scheduling.

The thread management in the middleware support is depicted in Fig. 1, which provides the heterogeneous computing resources (CPU and FPGA) for the execution flows. The threads can execute on either CPU or FPGA. Above the hardware infrastructure is the thread binding layer, where several effective binding algorithms are applied. Threads are created by linking the hardware/software libraries to the thread creation layer. The top layer is operating system library and user interface, which provide the multi-threads programming environment and integrate the designed hardware/software libraries.

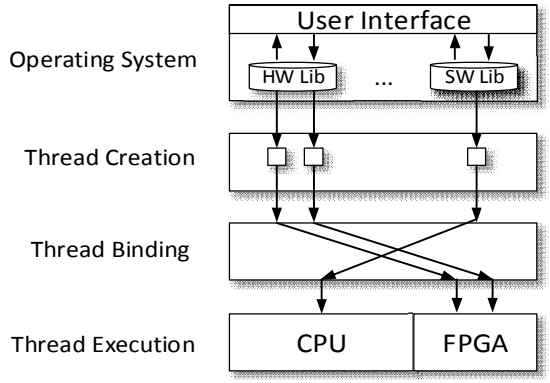


Fig. 1. Hierarchical Framework of Multi-Execution Flow

The hardware and software libraries are designed as dynamic linked libraries and integrated in the operating system. With linking to these libraries, traditional multi-thread program can run on either FPGA or CPU. Both software and hardware libraries contain a functions' entries table, which are ready to be invoked by user program. The difference between hardware and software libraries is that the hardware libraries integrate necessary ports and drivers to communicate with FPGA as well as the bit files to program the FPGA. The detail comparison between software library and hardware library is made in Table II. Furthermore, Fig. 2 illustrates the design and evaluation flow with the high level synthesis, which gives a comprehensive general processing flow for accelerator design in data-intensive applications.

TABLE II. Components in HW/SW Libraries

Items	Hardware Library	Software Library
Function Entries Table	Packaged hardware IP cores	Packaged functions
Drivers	Drivers for accelerators on FPGA	Not necessary
Synchronization Operations	In the **pthread_join** function	In the **pthread_join** function
Bit Files	Bit files for programming FPGA	Not necessary
Scripts Files	Script files for automatically reset and program FPGA	Not necessary

In order to be compatible with traditional multi-threads programs, the thread creation follows the POSIX thread standard. Programmers can use the any standard POSIX APIs, such as creating a thread, synchronizing operations, and terminating a thread. Basically, the hardware platform is transparent for programmers. A thread will be automatically issued to appropriate logic resources for execution.

We also provide an optional choice for users who need to manually map the thread to specified logic resource. There is a flag when creating the thread. The default value of the flag is NULL to indicate the framework will automatically map the thread to appropriate logic resources. If the programmers want their program run on the CPU, they could modify the flag to point to the corresponding entry in software library, and vice versa.

B. Out-of-order Scheduling with Inter-task Dependences

In the architecture, the system speedup is generally brought by the hardware accelerators. In order to enlarge the task level parallelism, we employ an out-of-order scheduling scheme with detection of inter-task dependences.

Inter-task data dependency is a key factor limiting the data-level parallelism. In this paper, we use sub-graphs as the fundamental unit during the data flow analysis step. Generally there are three types of data dependencies (including Write-after-Write, Read-after-Write and Write-after-Read). In order to make the analysis method general and not dependent on very specific application, we use Colored Petri Nets (CPN) for dependencies modeling and formal verification. In particular we abstract tasks running on heterogeneous reconfigurable system as instructions on uniprocessor as follows:

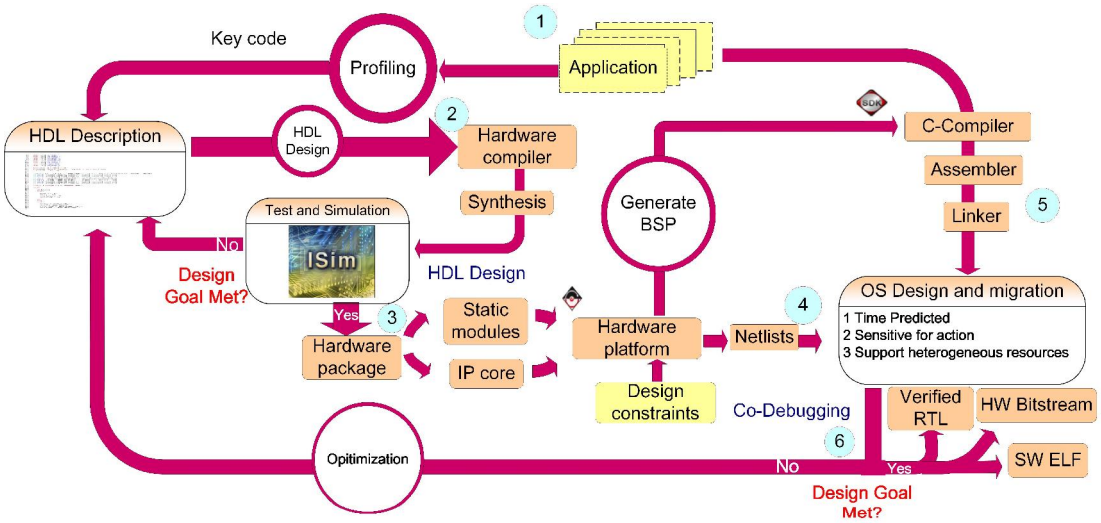
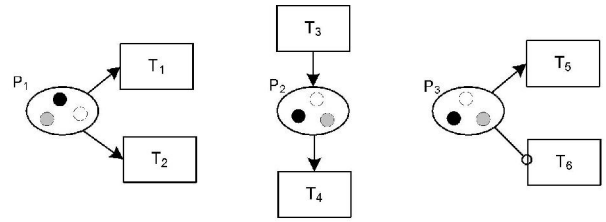


Fig. 2. Design & Evaluation Flow for High Level Synthesis

1) Abstract the task as instruction. The task type is abstracted as opcode instruction, while the input and output are abstracted as instruction operands.

2) Abstract the accelerator as functional unit. Each accelerator in the heterogeneous reconfigurable computing system is abstracted as a functional unit in Arithmetic Logic Unit (ALU), and in CPN model we use Place element to represent heterogeneous computing resources.



(a) Structural and Write-after-Write Data Dependencies (b) Read-after-Write Data Dependencies (c) Write-after-Read Data Dependencies

Fig. 3. Inter-task structural and data dependencies representations, P1~3 represents Place, and T1~6 represents Transition.

Fig. 3 illustrates the basic principle of the structural and data dependencies using CPN formal model. In Fig.3.a, there are three tokens in the Place P_1, which indicates the idle accelerators. The different colors means of these accelerators are heterogeneous. Transition T_1 and T_2 represent two computational tasks to run on the accelerators (denoted as Tokens in the Place P_1). If T_1 and T_2 need the same only token in P_1, then the trigger of either transition will consume the only token, leaving the other transition blocked immediately. In this manner, the structure dependency and Write-after-Write data dependencies could be formally described. Similarly, Fig.3.b and Fig.3.c depict the Read-after-Write and Write-after-Read data dependencies respectively. Please note the arc ended with a cycle Fig.3.c is the inhibitor arc, which means T_6 could be triggered only when Place P_3 is empty, which is corresponding to the Write-after-Read dependency. On the basis of these dependencies, we can get the partitioned tasks that could run in parallel to achieve data-level parallelism.

An out-of-order scheduling scheme is employed based on the inter-task data dependence, which is in charge of scheduling tasks to exploit the potential parallelism. Before the task can be issued, it will first detect the inter-task dependencies. If the current task does not depend on previous issued tasks, it can be issued immediately, otherwise it has to wait all the required input tokens to be ready. When the previous task finishes its execution, it will release the token and thereby the current task can be executed. This mechanism enables out-of-order for parallel threads running on hardware accelerators.

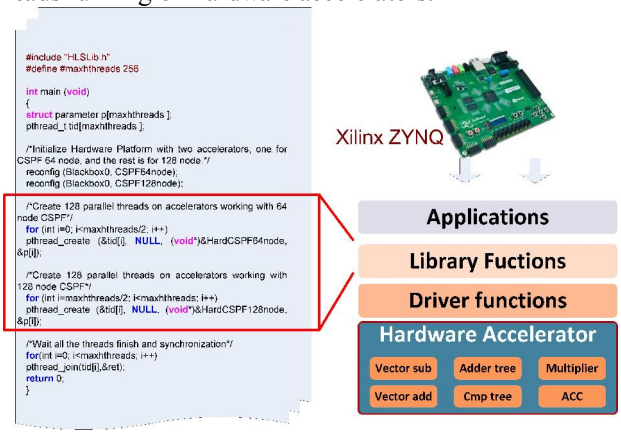


Fig. 4. Architecture Framework of the Prototyping System

IV. Experimental Platform and Results

In order to evaluate the prototype, we have built the general framework on the Xilinx state-of-the-art FPGA Zedboard platform, which integrates an ARM Cortex-A9 dual core processor with 667MHz frequency and a Zynq-7000 series FPGA fabric. The proposed hardware

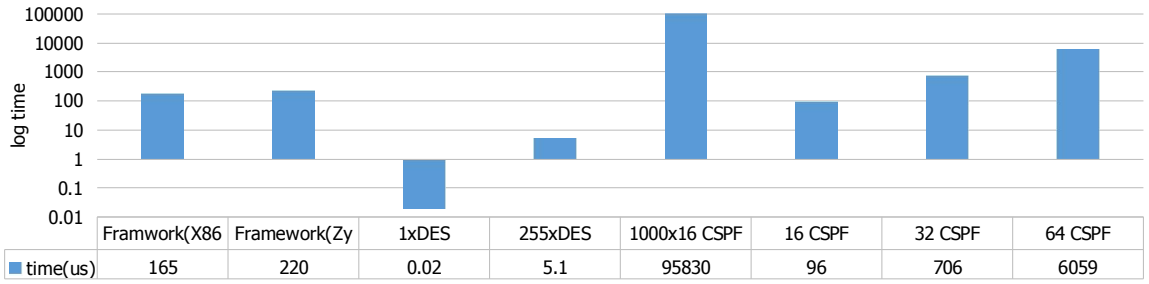


Fig. 5. Comparison of Framework overheads against typical benchmark applications.

architecture is presented in Fig. 4. The prototype consists of following layers: the application layer, the library function layer, the driver function layer, and the accelerator layer. The accelerator layer is composed of diverse hardware intellectual property (IP) cores as hardware acceleration engines, interconnect modules, local buses, memory blocks, and peripheral modules.

Figure 4 also illustrates the programming model on the experimental platform. In order to support hardware threads running on accelerators, we have implemented a kernel library to manipulate the process. The specific function will be abstracted as a thread, which will be offloaded to the accelerator for hardware execution. Based on the prototyping system, we evaluated *the overhead of the middleware, efficiency of the generated code (with/without optimization), and the running time of the RTL generation* respectively.

A. Overheads of the Middleware

With the help of the high level programming model and the middleware support, programmers can utilize the accelerators efficiently. However, overheads should be taken into consideration while offloading partial tasks into accelerators. We have evaluated the overheads of the framework, comparing with benchmark applications including Constrained Shortest Path Finding (CSPF) algorithm and DES applications. Experimental results in Fig.5 illustrate that the overheads of the platform on X86 (for comparison) and Zynq platform are 165us and 220us, respectively. It reveals that due to the overheads, it may not bring speedup for applications with low computational complexity, e.g. DES application less than 255 rounds, or CSPF application less than 32 rounds. In this situation, we can use the raw bare metal accelerators without any OS middleware. Meanwhile, when the applications become more complex, the overheads also become insignificant compared to the execution time on accelerators.

B. Efficiency of the RTL generated code

Fig.6 illustrates the comparison between "ideal" hardware execution time and generated time, counted by clock cycles. The "ideal" IP cores are achieved from OpenCores and supposed to be optimized. All the hardware IP cores runs at 100MHZ. It reveals that the execution time of the register transfer level (RTL) code generated from Vivado HLS is not even close to the execution time from the ideal IP cores, such as the FIR, DES and CSPF benchmarks. In comparison, the Matrix multiply application can achieve ideal speedup as it consists of regular calculation, on the condition of the loop unrolling and reshapge optimizations.

Furthermore, some results on quantitative efficiency derived from the comparison with optimization in Fig.6 is presented in Fig. 7. It reports that the optimization measures such as pipeline, unroll, reshape and memory reallocations can improve the efficiency of the generated hardware sources code, but not satisfying enough. For example, the

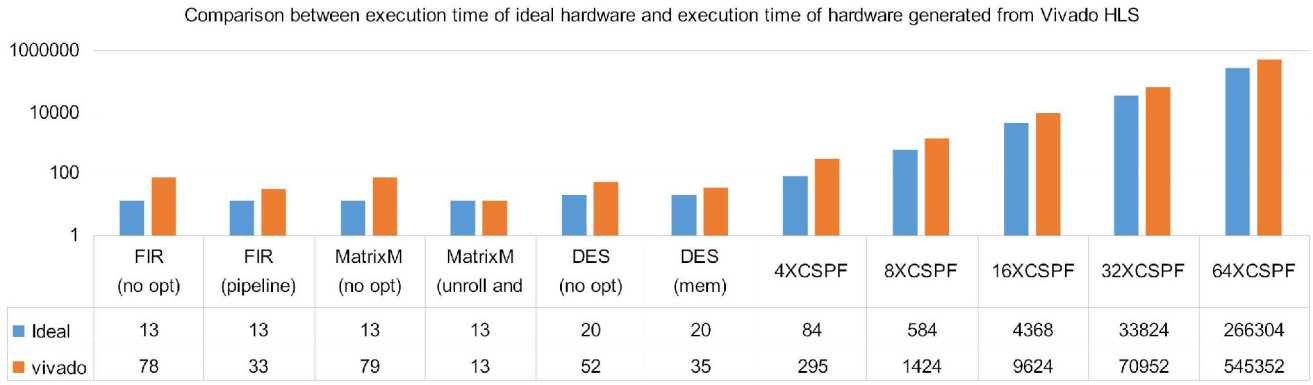


Fig. 6. Comparison between ideal hardware execution time and generated time

pipeline optimization technique has promoted the efficiency of FIR accelerators from 18% to 49% only, while the memory reallocations methods increase the efficiency of DES application from 38% to 57% approximately.

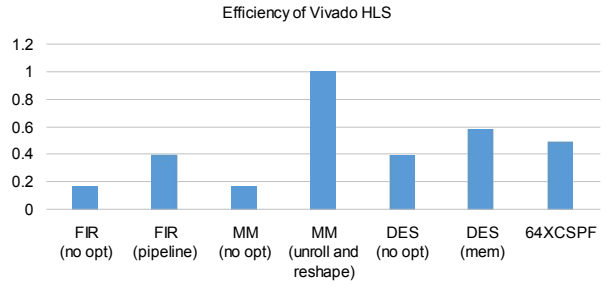


Fig. 7. Efficiency derived from typical benchmark applications

C. *Running Time of the RTL Generation*

Another perspective is the running time of the RTL generation process, which is illustrated in Fig. 8. We use CSPF as the case study, x-axis indicates the node number, and y-axis refers to the running time of the RTL generation (in Minutes). When the network contains nodes less than 32, the Vivado HLS tool is fast enough to generate the hardware code (several minutes). However, when the network has more than 128 nodes, it takes more than 4 hours to generate the RTL code, which means once the C code is modified, another long-time iteration will be taken into process.

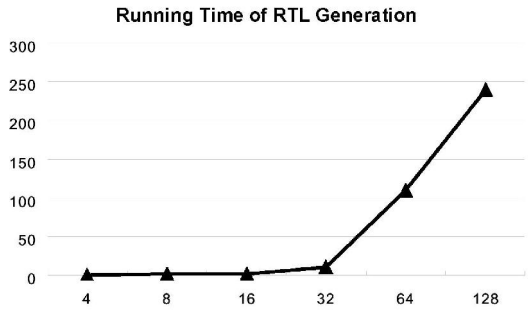


Fig. 8. Running time of RTL generation. The x-axis indicates the node number, and y-axis refers to the running time (denoted in Minutes).

V. CONCLUSIONS

To bridge the gap between high level software programmers and the hardware architecture, in this paper we have presented a hierarchical framework with HLS tools. Experimental results on Xilinx platform demonstrates the efficiency, optimization technique, and the running time of the RTL generation. Experimental results demonstrated the efficiency of the generated code is only 30%~50% for some typical applications. Also, the scheduling overheads of the OS middleware is significant enough to hide the speedup for some intuitive benchmark applications. Finally the running time of the software tools to generate RTL code shows that the optimization of the software tools is still worth pursing to facilitate software programmers.

REFERENCES

[1]. Howe D, Costanzo M, Fey P, Gojobori T, Hannick L, Hide W, et al. Big data: The future of biocuration. Nature. 2008;455(7209):47-50.

[2]. Chen T, Du Z, Sun N, Wang J, Wu C, Chen Y, et al. DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning. Proceedings of the 19th international conference on Architectural support for programming languages and operating systems; Salt Lake City, Utah, USA: ACM; 2014. p. 269-84.

[3]. Moreau T, Nelson MWJ, Sampson A, Esmaeilzadeh H, Ceze L, Oskin M, SNNAP: Approximate Computing on Programmable SoCs via Neural Acceleration. HPCA; 2015.

[4]. Liu F, Ghosh S, Johnson NP, August DI. CGPA: Coarse-Grained Pipelined Accelerators. 51st Design Automation Conference; San Francisco, CA, USA: ACM; 2014.

[5]. Yoshimi M, Nishikawa Y, Miki M, Hiroyasu T, Amano H, Mencer O, A performance evaluation of CUBE: one-dimensional 512 FPGA cluster. Proceedings of the 6th international conference on Reconfigurable Computing: architectures, Tools and Applications; 2010; Bangkok, Thailand: Springer-Verlag.

[6]. Tsoi KH, Luk W, Axel: a heterogeneous cluster with FPGAs and GPUs. Proceedings of the 18th annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays; 2010; Monterey, California, USA: ACM.

[7]. Shan Y, Wang B, Yan J, Wang Y, Xu N, Yang H, FPMR: MapReduce framework on FPGA. Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays; 2010; Monterey, California, USA: ACM.

[8]. Lin Z, Chow P, ZCluster: A Zynq-based Hadoop cluster. 2013 International Conference on Field-Programmable Technology (FPT); 2013 9-11 Dec. 2013.

[9]. Boyd-Wickizer S, Chen H, Chen R, Mao Y, Kaashoek F, Morris R, et al., Corey: an operating system for many cores. Proceedings of the 8th USENIX conference on Operating systems design and implementation; 2008; San Diego, California: USENIX Association.

[10]. Wentzlaff D, Agarwal A. Factored operating systems (fos): the case for a scalable operating system for multicores. ACM SIGOPS Operating Systems Review. 2009;43(2):76-85.

[11].Baumann A, Barham P, Dagand P-E, Harris T, Isaacs R, Peter S, et al., The multikernel: a new OS architecture for scalable multicore systems. Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles 2009.

[12].Andrews DL, Niehaus D, Jidin R, Finley M, Peck W, Frisbie M, et al. Programming models for hybrid FPGA-cpu computational components: a missing link. IEEE Micro. 2004;24(4):42-53.

[13].So HK-H, Brodersen RW. A unified hardware/software runtime environment for FPGA-based reconfigurable computers using BORPH. ACM Trans Embedded Comput Syst. 2008;7(2).

[14].Lubbers E, Platzner M. ReconOS: Multithreaded programming for reconfigurable computers. ACM Transactions on Embedded Computing Systems. 2009;9(1):1-33.

[15].OpenACC. Available from: http://www.openacc-standard.org/.

[16].Brown KJ, Sujeeth AK, Lee H, Rompf T, Chafi H, Odersky M, et al., A Heterogeneous Parallel Framework for Domain-Specific Languages. 2011 International Conference on Parallel Architectures and Compilation Techniques (PACT); 2011 10-14 Oct. 2011.

[17].Chafi H, Sujeeth AK, Brown KJ, Lee H, Atreya AR, Olukotun K, A domain-specific approach to heterogeneous parallelism. Proceedings of the 16th ACM symposium on Principles and practice of parallel programming 2011.

[18].OpenCL. Available from: http://www.khronos.org/opencl/.

[19].Etsion Y, Cabarcas F, Rico A, Ramirez A, Badia RM, Ayguade E, et al. Task Superscalar: An Out-of-Order Task Pipeline. Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture. 1935014: IEEE Computer Society; 2010. p. 89-100.

[20].Putnam A, Caulfield AM, Chung ES, Chiou D, Constantinides K, Demme J, et al., A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services. ACM/IEEE 41st International Symposium on Computer Architecture (ISCA); 2014.

[21] Chao Wang, Xi Li, Xuehai Zhou:SODA: software defined FPGA based accelerators for big data. DATE 2015: 884-887