SODA: Software Defined FPGA based Accelerators for Big Data

Chao Wang, Xi Li and Xuehai Zhou
School of Computer Science
University of Science and Technology of China
Suzhou, Jiangsu
{cswang,llxx,xhzhou}@ustc.edu.cn

***Abstract*—FPGA has been an emerging field in novel big data architectures and systems, due to its high efficiency and low power consumption. It enables the researchers to deploy massive accelerators within one single chip. In this paper, we present a software defined FPGA based accelerators for big data, named SODA, which could reconstruct and reorganize the acceleration engines according to the requirement of the various data-intensive applications. SODA decomposes large and complex applications into coarse grained single-purpose RTL code libraries that perform specialized tasks in out-of-order hardware. We built a prototyping system with constrained shortest path Finding (CSPF) case studies to evaluate SODA framework. SODA is able to achieve up to 43.75X speedup at 128 node application. Furthermore, hardware cost of the SODA framework demonstrates that it can achieve high speedup with moderate hardware utilization.**

Keywords—FPGA; Software-defined; Acceleratioin; Big data

I. INTRODUCTION

The cutting edge applications are becoming increasingly data-intensive, variable and fast changing. To meet the requirements of the novel big data era (*Volume*, *Variety*, *Velocity, Veracit* and *Value*), Field Programmable Gate Array (FPGA) has been employed as one of the most important solutions to accelerate key applications across from academic to industry, such as search engine in network [1], machine learning [2], and genome sequencing in bioinformatics [3]. For example, it has been proved that the integration of FPGA in Microsoft Bing search engine will accelerate the performance of kernel applications by more than 40X. As a consequence, the entire system throughput can be improved by 2.0X, while the budget and power consumptions of the datacenters can be significantly saved by 50% [1]. To extend the performance scaling on data-intensive applications in future parallel machines, how to improve the performance and energy efficiency has become a first-class priority.

Although accelerators have been demonstrated to be effective and efficiency at increasing performance, their benefits can also diminish given the scarcity of middleware support in operating system in the future. In the light of the parallel and distributed computer architectures, several studies have been conducted in the face of technology constraints, like Fos [4] and Barrelfish [5]. However, on the traditional opinion of designers, FPGA and GPGPU act as the accelerators of hardware that managed directly by the programmers, while the operating system and middleware afford no more support than conventional device drivers. This manner leads to a complicated design flow in the process of development. Furthermore, it is inefficient because of ignoring the potential parallelism of the individual logic blocks. The operating system must provide a considerable support for the reconfigurable device, so as to improve the utilization of FPGA chip and simplify the design flow.

Another problem we have to face is that the gap between high-level programmers and the hardware architect, which is very likely to narrow the application fields of the FPGA architectures, therefore it is essential and necessary to construct an efficient operating system support to enable various software defined accelerators [6] on FPGA based architectures.

In order to tackle the above problem, in this paper we propose Software defined accelerators SODA, which could manage the multicore system architecture according to the big data application behavior using FPGA platform, especially with heterogeneous accelerators. Unlike prior operating system using well-known thread/process execution model between simple and complex processors, Our concept must consider the less-obvious relationships between conventional processors and a diverse set of unconventional cores. SODA is targeted at heterogeneous architectures with the following characteristics:

1) Component based programming model: The unconventional heterogeneous accelerators are abstracted as software-defined components, which could facilitate researchers to incorporate heterogeneous functional modules easily, as well as to mitigate the burden of the programmers.

2) Dataflow execution: Instead of streaming and processing the entire dataset, the computation consists of dataflow execution phases, each of which continuously examines data subsets 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.

The remainder of this paper is organized as below. In Section II we summary the related work. Thereafter Section III details the SODA architecture, which includes the operating system framework, the programming model, and the design flow with out-of-order scheduling. Then in Section IV we

present the hardware prototype using Xilinx FPGA and Software Defined Network applications. Experimental speedup, overheads and hardware cost are illustrated. Finally, the paper is concluded in Section V.

II. Related Work

A. Architecture Design

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

Of the cutting-edge researches, the high-performance, low-power FPGA based computing mode has been renewed during the past few years. For example, [7] uses FPGA cluster system to accelerate IO-intensive streaming applications for efficiency. [8] utilizes FPGA-based cluster system to accelerate cellular automata simulation applications. Due to the application field of the above application-specific platforms, some researchers have proposed scalable FPGA cluster platforms, such as CUBE [9], Axel [10], FPMR [11], ZCluster [12]. CGPA [6] is a novel coarse grained accelerator based platforms on FPGA. However, these platforms still lack of effective program model and runtime support; therefore they could not be applied to all kinds of dataflow applications.

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 novel parallel multicore architectures. For example fos [4] and barrelfish [5]. fos [4] binds each service on an individual processor, to avoid cache competition and process switching. Barrelfish [5] abstracts the operating system a distributed cluster. Each processor has an asynchronous communication mechanism to the management modules via FIFO based interface.

To support heterogeneous reconfigurable technology, Hthread [13] presents a multi-threading model for FPGA based reconfigurable systems, the threads running on general-purpose processor are defined as software threads while the threads running on reconfigurable logic are defined as hardware threads. UC Berkeley's BORPH [14] stores the reconfigurable logic configuration information in the executable file, and utilizes FIFO buffer to achieve inter-thread communication. Since BORPH does not have the ability to schedule the reconfigurable resources, and thereby lack of support for dynamic reconfiguration features. ReconOS [15] is POSIX standard multi-threading operating system on a reconfigurable platform. The thread level communications are through shared memory interface, and the synchronization is based on semaphore mechanism.

III. SODA Architecture

A. SODA System Framework

The proposed hardware architecture of SODA is presented in Fig. 1. The architecture is constructed in a hardware platform that can provide heterogeneous reconfigurable multicore resources. It consists of following layers: an application layer, a kernel scheduling layer, a hardware abstraction layer, and several embedded processors as software computing nodes, various types of hardware intellectual property (IP) cores as hardware nodes, interconnect modules, processor local bus, memory, and peripheral modules.

1. Application layer is running on general purpose processor to provide the basic run-time environment and application programming interfaces (API) to tasks. Moreover, it also traces the application and sends all the tasks requests to scheduler for processing.

2. The middleware layer and the OS nodes are in charge of system virtualization, task partitioning, mapping, distributing and scheduling.

3. The hardware abstraction layer plays a key role to abstract hardware computing resources to OS. Both software and hardware communication units are employed to transfer partitioned tasks to either software nodes or hardware IP cores.

4. Software nodes: tasks can be distributed to either software or hardware nodes at run-time. All the software nodes are provided by general purpose processors. In general, this type of IP cores can run different kinds of tasks. Every software nodes holds a same functional library which includes all the available tasks of the whole system. Scheduler can decide and dispatch the tasks to different nodes considering current workload status of the system.

5. Hardware IP Cores: every hardware IP core is responsible for a specific task to achieve hardware acceleration. In addition, the hardware IP cores can be dynamically reconfigured according to application demands.

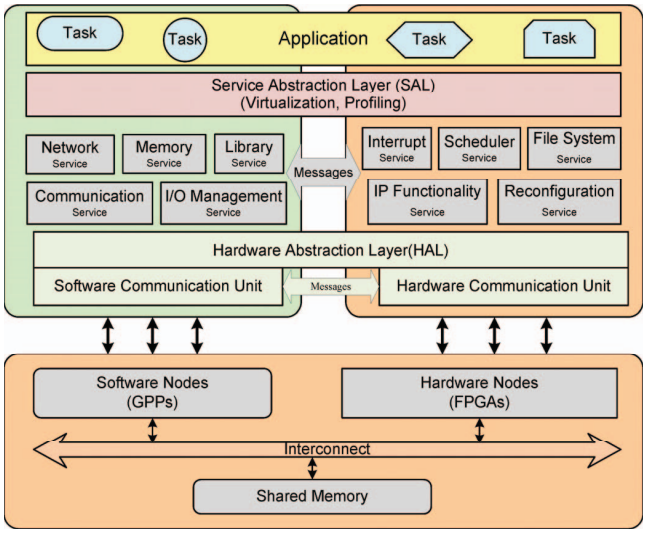

Fig. 1. Hierarchical Overview of SODA

B. Programming Model

The programming Model is extended from the Pthread based programming model. In order to support hardware thread running on accelerators, we have implemented a kernel lib to manipulate the hardware execution process. The programming model is extended from state-of-the-art Pthread models, which enables each hardware task execution as a specific thread. A code snippet using CSPF algorithm is listed in Figure 2.

```
#include "SODALib.h"
#define #maxhthreads 256

int main (void)
{
    struct parameter p[maxhthreads ];
    pthread_t tid[maxhthreads ];

/*Initialize Hardware Platform with two accelerators, one for CSPF 64 node, and
the rest is for 128 node.*/
    reconfig (Blackbox0, CSPF64node);
    reconfig (Blackbox0, CSPF128node);

/*Create 128 parallel threads on accelerators working with 64 node CSPF*/
    for (int i=0; i<maxhthreads/2; i++)
        pthread_create (&tid[i], NULL, (void*)&HardCSPF64node, &p[i]);
/*Create 128 parallel threads on accelerators working with 128 node CSPF*/
    for (int i=maxhthreads/2; i<maxhthreads; i++)
        pthread_create (&tid[i], NULL, (void*)&HardCSPF128node, &p[i]);

/*Wait all the threads finish and synchronization*/
    for(int i=0; i<maxhthreads; i++)
        pthread_join(tid[i],&ret);
    return 0;
}
```

Fig. 2. Code Snippet of Pthread based Programming Model

C. Design Flow and Scheduling

The design flow of SODA process is divided into following six steps.

1) The first step is to profile the big data application by the Hot Spot analysis method to find the key code. For steam applications, we need to explore the key step in the stream-like execution flow. Then the key code functions are ranked according to the priority.

2) Using the HDL description to design the hardware. Then the designed register-transfer code through the compiler and the hardware integration finally generates configuration stream file. Vivado high level synthesis tools can be utilized to generate RTL code. This file firstly runs in Xilinx ISim (etc. or ModelSim) simulation environment to verify the correctness of the behavior and timing.

3) If the verification can not meet the deign goals, we should redesign the HDL description. Till the verification is OK, we can package the designed hardware into IP cores. Here there are two main parts: one is a static module, also called fixed components platform, including microprocessors, memory, bus and peripherals. The other part is reconfigurable module, which is dedicated to encapsulated IP cores. We should integrate the two parts on the basis of meeting PGA design requirements and constraints, as well as generate Board Support Package (BSP) and Netlists. Netlist is used to generate the hardware bit stream, and BSP files can provide basic hardware description of the basic running environment.

4) After the hardware platform is constructed, a custom operating system is migrated into the hardware platform. The operating system can reconstruct and manage heterogeneous resources.

5) Compile the application: According to information about BSP, the application can be cross compiled, assembled and linked and finally generate an executable file. Then the operating system calls the designed hardware resources to run the application.

6) When the application is running, we should do collaborative debugging. If the result meets the design goals we can get the final design files, including hardware bitstream files, software executable ELF file for each application and the verified RTL implementation of specific functions. If the requirement is not satisfied, it should undergo a further hardware optimization and then design process.

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. The scheduler deals with the task sequence that sent to it, and there is no speculation execution. Based on the programming model described in the previous section, OoO task scheduler is implemented as a hardware layer to uncover task level parallelism. The scheduler dynamically detects and eliminates inter-task WAW and WAR dependencies, and thus speeds up the execution of the whole program. With the help of scheduler, programmers need not to take care of the inter-task data dependence.

IV. Experimental Results with Case Studies

In order to demonstrate the efficiency of SODA, we use Xilinx Zynq board to testify the experimental results using a CSPF algorithm in Software-Defined Network application. ARM Cortex A9 is equipped as main processor, which runs the SODA operating system framework, including the high level programming model, and the out-of-order scheduling. The basic architecture and interconnection is illustrated in [16]. One hardware accelerator is implemented at RTL and then integrated to accelerate the CSPF function.

A. Results and Discussion

We evaluated the SODA framework using CSPF algorithm in SDN applications, ranging from 4 nodes to 128 nodes. In order to evaluate the impact of the speedup as well as communication overheads, we measured the software execution time, hardware execution time, and calculate the speedup. The results depict that the SODA framework could accelerate the 128 node SDN application by 43.75X (when cache is closed). The curve also depicts that the speedup will continue to increase with the number of the node number of the network. The experimental results are illustrated in Fig.3.

Another interesting scope is the difference between achieved experimental speedup and the theoretical ideal speedup. The SODA presented in this paper could achieve more than 97.3% of the ideal speedup, which reveals that the scheduling overhead s loss is less 3%.

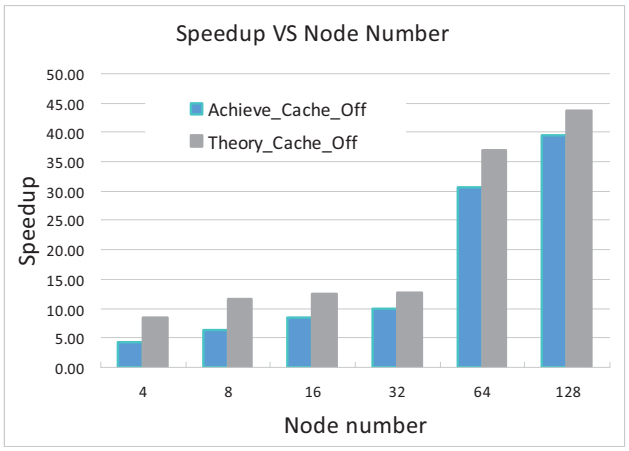

Fig. 3. Speedup of SODA in differnet node number

B. *Hardware Cost*

Our experimental platform is built on Xilinx Zynq FPGA. We have integrated the Microblaze processor, the CSPF accelerator, the bus channel, memory, and peripherals.

Table I compares the resources usage of the hardware scheduler with the ARM Cortex processor. It reports that the hardware scheduler only accounts 20.42% of the registers of ARM and 80.75% of the LUTs as Logic. The Total LUT of the hardware scheduler is 1668, while ARM takes 1518.

TABLE I. RESOURCES USAGE

	Hardware Scheduler	ARM Cortex
Registers	295(0.4%)	1445(2.1%)
LUTs used as Logic	1158(1.7%)	1434(2.1%)
LUTs used as RAM	510(0.7%)	84(0.1%)
Total LUTs	1668(2.4%)	1518(2.2%)

(*%): Resources used/Total on-chip resources

V. CONCLUSION

In this paper we have presented SODA, a software-defined operating system framework for FPGA based accelerators. SODA is composed of several layers. A programming model is given to facilitate researcher. Out-of-order scheduler with renaming techniques is proposed, which not only can detect the inter-task data dependence, but is also able to dispatch the hardware tasks into the accelerators in parallel. A case study with CSPF algorithm in SDN application demonstrates that SODA can efficiently manage the accelerators to achieve a significant speedup at 43.75X. Furthermore, the overhead of the programming model as well as the hardware cost is presented to demonstrate the low overheads of the SODA framework.

ACKNOWLEDGMENT

This work was supported by the National Science Foundation of China under grants (No. 61379040, No. 61272131, No. 61202053, No. 61222204, No. 61221062), Jiangsu Provincial Natural Science Foundation (No. SBK201240198), the Strategic Priority Research Program of CAS (No. XDA06010403), and Fundamental Research Funds for the Central Universities No. WK0110000034. The authors deeply appreciate many reviewers for their insightful comments and suggestions.

REFERENCES

[1].Andrew Putnam, Adrian M. Caulfield, et al. A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services. in ISCA. 2014.

[2].Tianshi Chen, Zidong Du, et al., DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning, in Proceedings of the 19th international conference on Architectural support for programming languages and operating systems. 2014, ACM: Salt Lake City, Utah, USA. p. 269-284.

[3].Chao Wang, Xi Li, et al., Heterogeneous Cloud Framework for Big Data Genome Sequencing. IEEE/ACM Transactions on Computational Biology and Bioinformatics, 2014. PP(99): p. 1.

[4].David Wentzlaff and Anant Agarwal, Factored operating systems (fos): the case for a scalable operating system for multicores. ACM SIGOPS Operating Systems Review, 2009. 43(2): p. 76-85.

[5].Andrew Baumann, Paul Barham, et al. The multikernel: a new OS architecture for scalable multicore systems. in Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles 2009: p. 29-44

[6].Feng Liu, Soumyadeep Ghosh, et al., CGPA: Coarse-Grained Pipelined Accelerators, in 51st Design Automation Conference. 2014, ACM: San Francisco, CA, USA.

[7].Andrew G. Schmidt, Siddhartha Datta, et al., Investigation into scaling I/O bound streaming applications productively with an all-FPGA cluster. Parallel Computing, 2012. 38(8): p. 344-364.

[8].S. Murtaza, A. G. Hoekstra, et al., Cellular Automata Simulations on a FPGA cluster. International Journal of High Performance Computing Applications, 2011. 25(2): p. 193-204.

[9].Masato Yoshimi, Yuri Nishikawa, et al. A performance evaluation of CUBE: one-dimensional 512 FPGA cluster. in Proceedings of the 6th international conference on Reconfigurable Computing: architectures, Tools and Applications. 2010. Bangkok, Thailand: Springer-Verlag: p. 372-381.

[10].Kuen Hung Tsoi and Wayne Luk. Axel: a heterogeneous cluster with FPGAs and GPUs. in Proceedings of the 18th annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays. 2010. Monterey, California, USA: ACM: p. 115-124.

[11].Yi Shan, Bo Wang, et al. FPMR: MapReduce framework on FPGA. in Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays. 2010. Monterey, California, USA: ACM: p. 93-102.

[12].Zhongduo Lin and Paul Chow. ZCluster: A Zynq-based Hadoop cluster. in 2013 International Conference on Field-Programmable Technology (FPT). 2013: p. 450-453.

[13].David L. Andrews, Douglas Niehaus, et al., Programming models for hybrid FPGA-cpu computational components: a missing link. IEEE Micro, 2004. 24(4): p. 42-53.

[14].Hayden Kwok-Hay So and Robert W. Brodersen, A unified hardware/software runtime environment for FPGA-based reconfigurable computers using BORPH. ACM Trans. Embedded Comput. Syst., 2008. 7(2).

[15].Enno Lubbers and Marco Platzner, ReconOS: Multithreaded programming for reconfigurable computers. ACM Transactions on Embedded Computing Systems, 2009. 9(1): p. 1-33.

[16]. Chao Wang, Xi Li, et al., Architecture Support for Task Out-of-order Execution in MPSoCs. IEEE Transactions on Computers, 2014.