Exploiting Hardware Reconfigurability on Window Join

Eric Shun Fukuda*, Hideyuki Kawashima†, Hiroaki Inoue‡, Tetsuya Asai* and Masato Motomura*
* Graduate School of Information Science and Technology
Hokkaido University, Sapporo, Hokkaido 060-0814, Japan
fukuda@lalsie.ist.hokudai.ac.jp
† Faculty of Systems and Information Engineering
University of Tsukuba, Tsukuba, Ibaraki 305-8577, Japan
‡ NEC Corporation
Kawasaki, Kanagawa 211-8666, Japan

POSTER PAPER

Abstract— **Stream processing is attracting wider attention in recent years, and in order to get high efficiency, more people are now trying to leverage hardware for stream processing. In this paper, we clarify two issues by taking window join as an example application: a) how a software engineer would efficiently utilize hardware, and b) how adaptiveness will be achieved on it. We use a dynamically reconfigurable hardware with a C-based high level synthesis tool as our evaluation platform. The throughput improved by 216 times through software code optimization, and achieved 26 times higher throughput/power efficiency than an optimized software solution for a CPU. We conclude that a software engineer with certain hardware knowledge will be able to facilitate hardware, and dynamic reconfiguration capability improves the throughput/power efficiency of stream processing.**

Keywords- stream processing; window join; high level synthesis; processor architecture

I. INTRODUCTION

Stream processing is attracting considerable attention [1] as an important computation paradigm in the era of big data and cloud computing. Although they are dealt with distributed processors on parallel servers, it is expected that the demand for higher throughput and the power consumption will still continue to grow. In view of solving this issue, hardware-oriented acceleration of stream processing using field-programmable gate arrays (FPGAs) has been actively studied [2]-[4]. In essence, hardware customized to a given problem can achieve much higher throughput/power than a software solution that runs on general-purpose hardware. However, such hardware solutions typically have two major drawbacks: (1) they have limited in-field flexibility and (2) software engineers find it difficult to design them. As *adaptive query* becomes an important notion in stream processing, issue (1) needs to be addressed seriously, and since the algorithms are mainly developed by software engineers, (2) is important too. Solving these problems on an FPGA-based framework has been an active research topic in recent years. However, the dynamically reconfigurable processor (DRP) developed by one of the authors of this paper [5] may serve as another good foundation to overcome these drawbacks due to its outstanding in-field flexibility and its C-based design environment.

This work was partially supported by JSPS Grant-in-Aid for Challenging Exploratory Research (No. 24650033).

We carried out an experimental step-by-step (eight step) implementation of adaptive stream processor on DRP by considering window join (a stream version of SQL join), a simple but extensively studied important operation in stream processing, as a case study. Our goal is to find answers to the following questions: i) how a software code for hardware synthesis should be optimized, ii) how performance changes according to different modifications, iii) how dynamic reconfiguration helps achieving adaptiveness of the solution.

II. RELATED WORK

One of the works on hardware-accelerated stream processing uses a C-based high-level synthesis tool [2]. This system allows users to specify complex events by sequencing simple events described with custom C functions that will be synthesized into hardware. Another work uses SQL queries for describing the algorithm [3] whose operators are mapped to corresponding tiny circuit elements. Both of these works are further studied to be dynamically reconfigurable [4] [6]. Since DRP is inherently dynamically reconfigurable and has a C-based design tool, it should have the potential to be the platform for stream processing.

III. IMPLEMENTATION

The feature of each step was as follows:

1. Simple C code that software engineers would probably

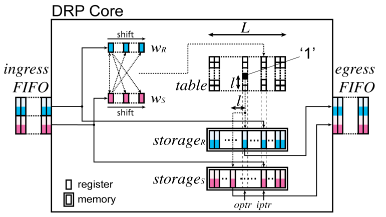

Figure 1. Synthesized architecture of Step 7.

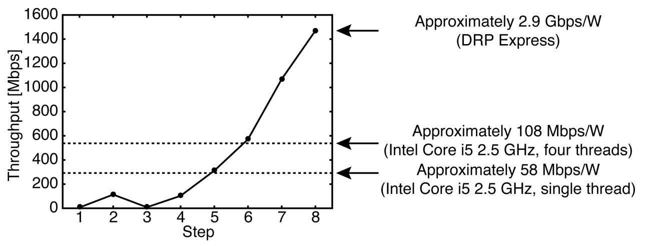

Figure 2. Improvement of the throughput.

write as their first step.

2. Sliding window buffer which reduces the number of memory accesses for fetching the incoming data.
3. Parallelized output data buffer which enables comparisons between two stream data to be done in parallel.
4. A bit table that holds the comparison results and as a result outplaces the register-consuming parallel output buffers.
5. Prefetching of the incoming data in chunk which reduces the delay of incoming data.
6. Accessing the table in parallel by dividing the 32-bit wide table into 8-bit width. (DRP has 8-bit architecture.)
7. Folding (pipelining) of the main loop.
8. Low match optimization by separating the output procedure which scarcely runs from the main loop.

Figure 1 shows the synthesized architecture of Step 7. It shows the sliding window buffer in the left and the table in the upper right. All steps were implemented in C, including the folding feature which could be implemented simply by specifying a folding attribute to a loop.

IV. Discussion

Figure 2 highlights the throughput improvement through the optimization. It indicates that the final step is 216 times faster than the first step. For reference, window join runs on an Intel Core i5-2520M at 539 Mbps when it runs in four threads. The relative ratio of the throughput/power of the DRP to that of the CPU is 26. This further proves the advocated throughput/power advantage of reconfigurable hardware acceleration.

The throughput once falls in Step 3 because it requires a lot of registers for synthesizing the code in Step 3. Figure 3 shows the efficiency of the resource usage and analyzes the implementation process in more details than [7]. In the figure you can see that the resources are not efficiently used in Step 3. Therefore in Step 4, a new architecture was introduced in order to substitute the register usage with memories. The efficiency of the resource usage increased greatly in the following steps which in consequence improved the throughput. Although the efficiency of memory usage reduced in Step 8, it helped the

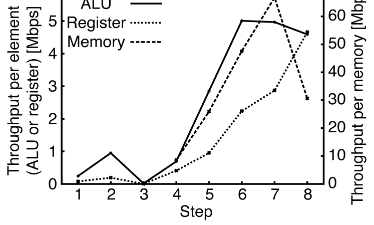

Figure 3. Efficiency of resource usage.

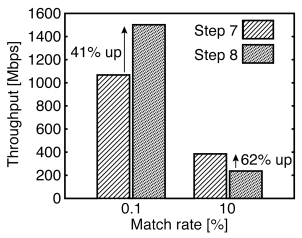

Figure 4. Match rate dependency (Step 7 vs 8).

efficiency of register (it is a faster resource than a memory) usage increase.

Figure 4 compares the throughput of Steps 7 and 8, the latter being optimized for a low match rate between the tuples in two streams. Since the DRP core can reconfigure its configuration cycle-to-cycle, it can switch its architecture between Steps 7 and 8, and thereby operate more efficiently than a non-dynamically reconfigurable hardware.

V. Conclusion

Our research motivation was to address two issues a) how and how much a software engineer can cultivate hardware acceleration, and b) how adaptiveness can be achieved in such a solution. We conclude that I) a state-of-the-art high-level synthesis tool is sufficiently powerful for writing all source code in C, however II) software engineers using the tool should still have some knowledge of hardware development and III) the dynamic reconfiguration of DRP provides a good means for adaptive processing. As future work, we will examine our proposal on various platforms such as commercial FPGA.

Acknowledgment

We are deeply grateful to Koichiro Furuta, Taro Fujii, Takeshi Inuo, and Takao Toi at Renesas Electronics Corporation for their helpful discussion and support.

References

[1] A. Arasu, S. Babu, and J. Widom, "The CQL continuous query language: semantic foundations and query execution," The VLDB Journal, vol.15, no.2, 2006.

[2] H. Inoue, T. takenaka, and M. Motomura, "20Gbps C-based complex event processing," Proceedings of the 2011 21st International Conference on Field Programmable Logic and Applications (FPL), 2011.

[3] R. Muller, J. Teubner, and G. Alonso, "Stream on wires – a query compiler for FPGAs," Proceedings of the VLDB Endowment, vol.2, no.1, 2009.

[4] M. Takagi, T. Takenaka, and H. Inoue, "Dynamic query switching for complex event processing on FPGAs," Proceedings of the 2012 22nd International Conference on Field Programmable Logic and Applications (FPL), 2012.

[5] M. Motomura, "A Dynamically reconfigurable processor architecture," Microprocessor Forum, 2002.

[6] T. Miyoshi, H. Kawashima, Y. Terada, and T. Yoshinaga, "A coarse grain reconfigurable processor architecture for stream processing engine," Proceedings of the 2011 21st International Conference on Field Programmable Logic and Applications (FPL), 2011.

[7] E.S. Fukuda, H. Kawashima, H. Inoue, T. Fujii, K. Furuta, T. Asai and M, Motomura, "C-based adaptive stream processing on dynamically reconfigurable hardware: a case study on window join," to appear in Proceedings of the 2013 9th International Symposium on Applied Reconfigurable Computing (ARC), 2013.