A Restricted Dynamically Reconfigurable Architecture for Low Power Processors

Takeshi Hirao Dahoo Kim Itaru Hida Tetsuya Asai Masato Motomura

Hokkaido University, Graduate School of Information Science and Technology,
Sapporo, Hokkaido, 060-0814, Japan

Abstract— In this paper, we propose a Control-flow Driven Data-flow Switching variable datapath architecture for embedded applications that demand extremely low power consumption and a wide range of usage. In the proposed architecture we aim to achieve both flexibility and low power consumption by limiting the scope of dynamic reconfiguration. As a preliminary evaluation, we have mapped a small program to understand the fundamental characteristics of the proposed architecture.

I. Introduction

In recent years low power consumption has been one of the main criteria governing the design of various sensor network and mobile processor. As an approach that aims for high performance/power of the processor, there are dynamically reconfigurable processor (DRP) and the reconfigurable processor. DRP execute large programs in time division manner (i.e., dynamic reconfiguration) and need to reconfigure variable datapath that configured on the switch and Processing Element(PE) at runtime . So, If there is frequently reconfigurations, Muccra[1], DRP[2] are conventional DRPs that require more power for dynamic reconfiguration.

In this study, we propose Control-flow Driven Data-flow Switching(CDDS) variable datapath architecture with the aim to improve the performance/power for applications, such as including the frequent switching of the control flow. By simultaneously maintaining the flexibility for the branch, and limiting the scope of the dynamic reconfiguration as possible, CDDS architecture aims to improve the performance/power.

II. CDDS Architecture

When a programs is represented in a control/data flow graph, a set of data flows to be actually executed is selected at a branching point in the control flow. Here, one can notice that the data flow actually affected at this point is the connection between the executed data flow and the selected data flow. This insight leads us to a new architecture, where a variable datapath is divided into a fixed part and a variable part. The fixed part runs only combinatorially, depending on the data dependency between instructions. Only the variable part is reconfigured at run time when a control flow branches are encountered. The name CDDS came from its intrinsic characteristic, where only the minimum portion of data flow switches under the supervision of a control flow. Since the scope of dynamic reconfiguration is quite limited compared to conventional dynamically reconfigurable processors, there is a good chance for this architecture to achieve both versatility and performance/power improvement fairly well.

A control/data flow graph of a program is shown in Fig.1. This program first run from Basic Block1(BB1) to BB2. Since there is no branch in data flows in BBs, so the data flows are not switched at runtime. Then depending on the result of the branch, data flows between BBs are switched. For example, when the branch is taken, data flow from BB2 to BB2 is needed as shown in Fig.1(b). On the other hand, when the branch is not taken, the data flow from BB2 to BB3 is required as shown in Fig.1(c).

According to the concept explained above, this program is then divided and mapped into two parts in the CDDS architecture: a fixed part and a variable part as shown in Fig. 2. All the intra-BB data flows are extracted and configured into the fixed part. Other inter-BB data flows are also extracted and configured into multiple contexts for the variable part. The variable part is reconfigured according to a context during execution of the task, and provides data to the BBs mapped on the fixed part according to the selected control flow. For example, context 2 and context 3 provides data connection from BB2 to BB2 and BB2 to BB3, respectively (see Figs. 1 and 2). Context switch is controlled by a state transition function generated from the control/data flow graph. Here, single branch generates two contexts (one for taken and the other for not taken). Since there is always an initial context, total number

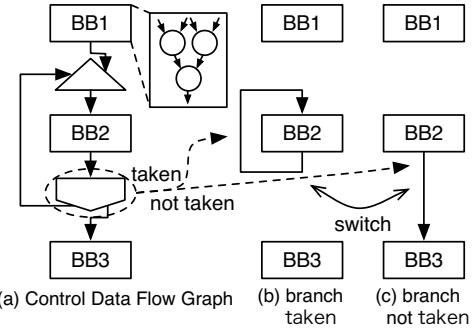

Fig. 1. Switching data flow between BBs

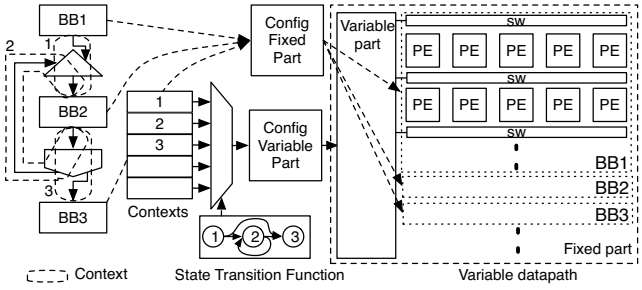

Fig. 2. The CDDS architecture concept: dividing a hot path into fixed part and variable part.

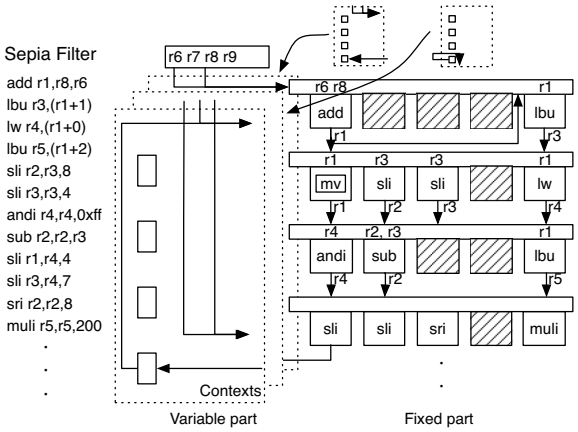

Fig. 3. sepia filter mapped on the CDDS variable datapath for evaluation.

of contexts for a given task is 2n+1, where n is the number of branches in the task. Thus, by dynamic reconfiguration of some parts, it is possible to achieve versatility for the branch instruction. By the accelerator of this CDDS architecture placing beside a base processor and executing hot path, we aim at a high performance/power.

III. Preliminary Evaluation Results

This chapter describes the preliminary evaluation results of the CDDS variable datapath. As a evaluation application, we used sepia filter to be used widely in benchmark. We first used the development environment for Lattice Mico32(LM32)[3] of Lattice Semiconductor Corporation, and compiled a program to generate a binary code. We then disassembled the binary code, and manually generated the configuration information. This configuration information was mapped on the CDDS variable datapath, as shown in Fig.3. We assigned instructions to PEs in the fixed part. Since the fixed part is not switched. On the other hand, the variable part is switched. Because there is one branch instruction in sepia filter, the number of contexts is 3. Hatched PEs is parts can not be mapped instructions. PE utilization of sepia filter is 57%. PE utilization is determined from the diagram mapped instruction sequence by dividing the number of PE mapped instructions by the total number of PE.

Next, we determined parallelism by dividing the execution time of LM32 by the execution time of the CDDS variable datapath. LM32 execution time was calculated as LD/ST, multiplication instruction is 2 cycle, and the other instruction is 1 cycle in each contexts. In addition, from the figure of mapping the instruction sequence, the execution time of the CDDS variable datapath was calculated by a part that was the longest chain in each context.

The mapping results of sepia filter are shown in Table I. Context 1 and, 3 are run only once, context 2 is repeated image size times. Image size is large enough, so parallelism of programs is parallelism of Context 2, which is 1.9.

PE utilization and parallelism is low because the data dependencies are high in program, and LD/ST is not run in parallel. However, PEs are connected combinatorially using switches without passing through registers; therefore, the results of the PE processing are completed faster, the delay time is shorter.

TABLE I
Mapping results on the CDDS variable datapath.

Context number	LM32 execution time	CDDS execution time	parallelism	Context num of executions	PE utilization
1	27	14	1.9	1	57
2	27	14	1.9	Image size	
3	1	1	1	1	

IV. Conclusion

We have proposed a CDDS architecture that simultaneously maintained flexibility, and also aimed at improving the performance/power. The CDDS architecture is characterized by following features: a datapath is divided into fixed and variable part, and only the variable part is reconfigured at branch points.

In this paper we described the idea and architecture of CDDS. We are now executing detailed RTL design of this architecture and will emulate it on an FPGA platform in near future, which should be published elsewhere.

Acknowledgements

This work was partly performed in collaboration with Semiconductor Technology Academic Research Center.

References

[1] Yoshiki Saito, Toru Sano, Masaru Kato, Vasutan Tunbunheng, Yoshihiro Yasuda, and Hideharu Amano.: *A real chip evaluation of muccra-3: A low power dycamically reconfigurable processor array*, In *ERSA'09*, pages 283–286, 2009.

[2] Masato Motomura.: *A dynamically reconfigurable processor architecture*, *Microprocessor Forum, Oct. 2002*, 2002.

[3] LatticeMico32 development tools, `http://www.latticesemi.co.jp/products/designsoftware/micodevelopmenttools/index.cfm`.