A HIGH SPEED DESIGN AND IMPLEMENTATION OF DYNAMICALLY RECONFIGURABLE PROCESSOR USING 28NM SOI TECHNOLOGY

Toru Katagiri and Hideharu Amano

Dept. of ICS, Keio University, Yokohama Japan
email: wasmii@am.ics.keio.ac.jp

ABSTRACT

Although dynamically reconfigurable processor arrays (DRPAs) are advantageous for embedded devices because of their high energy efficiency, many of the recent mobile devices are required to execute increasingly performance-centric jobs. One fairly straingtfoward way of increasing the clock frequency is introducing a pipelined structure into each PE. However, this results in frequent pipeline stalls due to the data hazard between multiple PEs.

In order to mitigate the effect of data hazard between PEs, we propose a tiny vector instruction mechanism. With a single vector instruction, a small amount of data is continuously processed in the pipeline of the PE. Pipeline stalls are removed without increasing the number of hardware contexts, and thus the amount of configuration data. Evaluation results based on the implementation using 28nm SOI process technology, a DRPA with tiny vector instructions (DRPA-TVI) improves the performance by 2.4 three times compared to a base DRPA with just a small increase of area and power consumption.

1. INTRODUCTION

Coarse-grained dynamically reconfigurable processor arrays (DRPAs) have been attracting attention as flexible and power-efficient accelerators for multimedia applications on system on chip (SoC) circuits. Some of these devices are now commercially available[1][2], and others have been the subject of various academic research.

A typical DRPA consists of processing elements (PEs) for calculation, local data memories, context memories, and switching elements (SEs) for interconnection of PEs. Here, context means a set of configuration data for operations and the interconnection of the PE array to be pre-loaded before running the applications. The behavior of PEs and the connection between PEs are dynamically reconfigured in accordance with hardware contexts read from context memories, and dynamic reconfiguration can be performed in either one or several clock cycles. DRPAs with such a mechanism are called multi-context DRPAs. By preparing contexts for each target application, DRPAs can execute various types of processing, unlike fixed hardware. For example, in ASICs consisting of hardwired logic, not the whole circuit but only one part usually works. By dividing the processing of the application into contexts, DRPAs can perform the whole function of an ASIC on a small semiconductor area without much performance degradation. DRPAs are thus advantageous in terms of development cost, area, and power consumption.

In recent years, a performance enhancement of DRPA is required along with the increasing demand for versatile mobile devices. One way to achieve a higher performance on DRPAs is to increase the number of PEs in the array. However, the utilization of PEs is limited depending on the application, and relying on highly parallel processing is not always the best solution.

The simplest method is to improve the operational speed of PEs rather than increasing their number. Although using pipelined PEs helps increase the operational frequency of DRPAs, it is difficult to fill all stages of a number of pipelined PEs. We propose in this study, tiny vector instructions into DRPAs to prevent stalls in the pipelined PEs with a limited number of contexts. Then, we show a design example using 28nm SOI technology with the evaluation results on area, performance and energy consumption.

2. INTRODUCING PIPELINED PE STRUCTURE

2.1. Base DRPA Architecture

The base DRPA used in this paper is referred to as DRPA-base. It is designed based on MuCCRA-3 [3] a prototype research DRPA except the structure of both interconnections. In order to increase the operational frequency, island-style network in MuCCRA-3 is changed into a point-to-point interconnection in DRPA-base.

2.2. Pipelined PE

The simplest way to increase the operational frequency is introducing a pipelined structure. As with conventional processors, four steps are needed to execute arithmetic oper-

The authors thank to VDEC, STARC, CMT, STmicro, Cadence and Synopsys for their cooperation of chip development.

ations in the PE: instruction fetch (IF), instruction decode (ID), execution (EX), and write back (WB). The process is divided by inserting pipeline registers between each stage (IF0, IF1, ID, EX, and WB) so that a standard five-stage pipeline can be built.

2.3. Pipeline Hazard and Related Work

The operating frequency is improved by introducing a pipeline structure to the PE. However, the pipelined PE structure is not commonly used in DRPAs because it is substantially unsuitable for its control mechanism. In DRPAs, the results of a PE are utilized by other PEs, which create data dependency between PEs. This means there are a lot of data hazards between the pipelines of multiple PEs and the accompanying risk of stalls. In DRPAs, all instructions are fetched from the context memory according to the common context pointer distributed from the controller, and once the pipeline of a PE stalls, it will cause all the PEs in the array to stall. Frequent instruction stalling thus occurs, which virtually kills any benefit provided by the pipelining. Out-of-order execution of context, which corresponds to the out-of-order instruction execution in conventional processors, is not a realistic solution since almost no hardware contexts can have their execution order changed.

For this reason, pipelined PEs are not so popular in multi-context DRPAs. In PipeRench [4], a row of PEs form a pipeline stage on which pipeline operation is performed with the whole PE array. The operation of the stage is reconfigured clock by clock, which allows for the implementation of a virtual long pipeline. In other DRPAs, pipeline operation is adopted between PEs globally rather than inside the PE locally.

3. TINY VECTOR INSTRUCTIONS

3.1. Introducing Vector Instructions

Here, we propose using vector instructions in the pipelined structure in order to treat multiple pieces of data with one instruction. By iteratively executing a single instruction to multiple data elements, stalls in the pipelined PEs can be eliminated without increasing the number of hardware contexts. In general, vector instruction is effective for processing with loop-level parallelism such as multimedia application, which is one of the main target applications of DRPAs. However, the pipeline depth of the DRPA-base is much shallower than that of the conventional vector processor [5] [6], which uses floating point instructions. Moreover, introducing vector registers will increase the total amount of hardware in the PE array. Therefore, we introduce tiny vector instructions just for keeping the PE pipeline full. Unlike with conventional vector instructions, only a small number of data elements are the target of the vector processing.

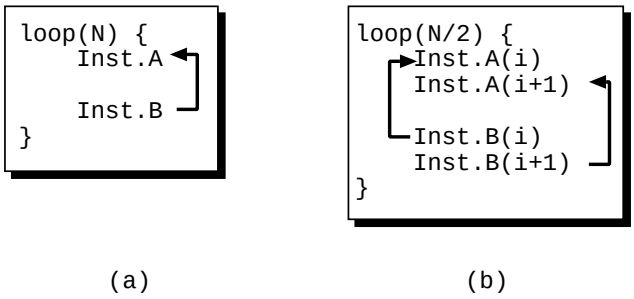


Fig. 1. Example of tiny vector instructions.

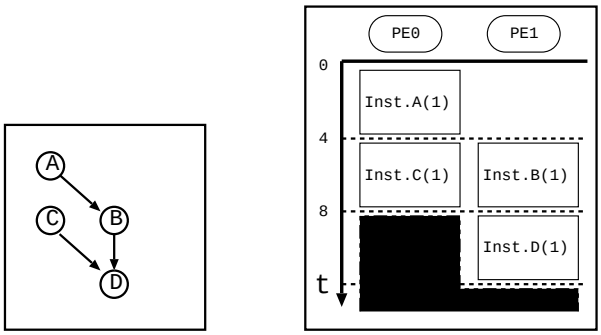
As an example to explain how this technique works, a cord with data dependence is shown in Fig. 1(a), with data dependence existing between Inst.A and Inst.B. Conventionally, in order to ensure proper execution on a pipelined PE, it would be necessary to create a stall (or insert an NOP instruction) between Inst.A and Inst.B.

In contrast, by using tiny vector instruction, the processing is performed automatically, similarly to scheduled loop unrolling, as shown in Fig. 1(b). The stall (NOP) between Inst.A and Inst.B is eliminated, since it is possible to keep a distance between them that has data dependence, indicated by the arrows in the figure. Therefore, the PE can execute the instructions efficiently. Although it is also possible to perform the same processing as Fig. 1(b) by rewriting the program explicitly, as with loop unrolling in conventional processors, the number of contexts, i.e., the configuration data, is doubled. Unlike conventional processors, which can use an enormous number of instructions, the number of hardware contexts in DRPAs is limited to from 16 to 64. Here, the number of vector elements is fixed as two and vector execution is specified as a bit attached to an instruction. Thus, almost no overhead is required for introducing tiny vector instructions. From here, we call DRPA-PL with such tiny vector instructions DRPA-TVI.

3.2. Example of Scheduling and Mapping

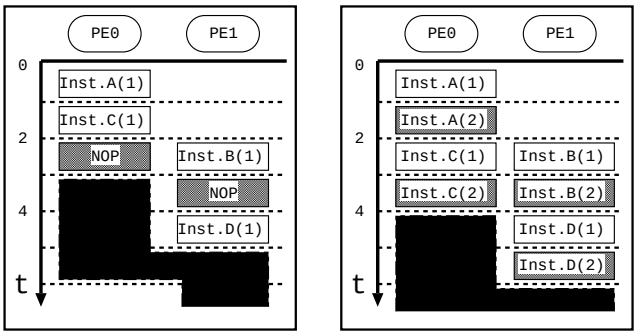
Figure 2(b), (c), and (d) show how instructions are scheduled and mapped on the PEs of DRPA-base, DRPA-PL, and DRPA-TVI, respectively, when the data flow graph shown in Fig. 2(a) is processed repeatedly in two PEs.

On DRPA-base, instructions can be scheduled in a form similar to the data flow graph shown in Fig. 2(b). In contrast, on DRPA-PL, it is necessary to stall (insert NOP), as shown in Fig. 2(c), to accommodate the data dependence. Consequently, for every loop, there are two instructions that are not effectively used. DRPA-TVI can use pipelined PEs effectively by executing the same instruction twice and maintaining the distance between instructions with data dependence. In addition, the instructions of the second iteration (indi-



(a) data flow graph

(b) Mapping on DRPA-base



(c)Mapping on DRPA-PL

(d) Mapping on DRPA-TVI

Fig. 2. The scheduling and mapping of instructions.

cated by shading in the figure) do not have to be prepared as configuration data. These instructions are automatically executed, and so the same configuration data for DRPA-base can be used in DRPA-TVI.

4. EVALUATION

4.1. Implementation using 28nm SOI

DRPA-TVI was designed using ST micro's 28nm SOI process technology. The design was described in Verilog-HDL, and Cadence NC-Verilog was used for simulation. Synthesis and layout were done with Synopsys Design Compiler and IC-Compiler. Since the usable area is limited to just 0.8mm square, only a small PE array of 2×2 was implemented. Empty area of the left side was used to test other circuits, an inductive coupling through chip interface. Although the size of PE array and the number of I/O were strictly limited, the pipeline structure and TVI mechanisms were fully implemented.

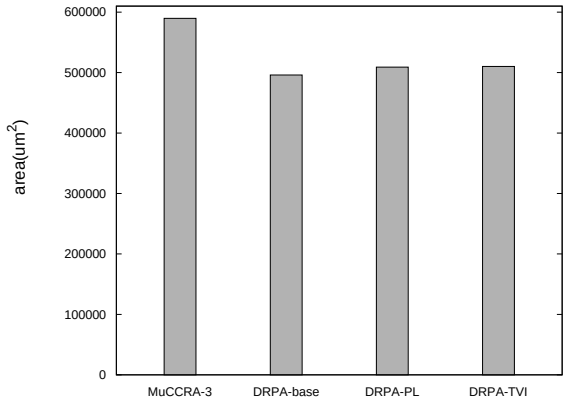


Fig. 3. Evaluation results: Area.

Table 1. Operational frequency.

Architecture	Frequency
MuCCRA-3	344MHz
DRPA-base	606MHz
DRPA-PL/DRPA-TVI	1.47GHz

4.2. Area and clock frequency

We assumed area enough to implement a PE array of 4×4, and layou-ted DRPA-base, DRPA-PL, and DRPA-TVI. For comparison, we implemented MuCCRA-3[3] with the same process. As described before, MuCCRA-3 uses a combination network with the island style and direct interconnection. The structure of the PE is the same as DRPA-base. Figure 3 shows the area of MuCCAR-3, DRPA-base, DRPA-PL, and DRPA-TVI. The area is represented with the total amount of cell area.

Compared to DRPA-base, the area of DRPA-PL and DRPA-TVI increased gradually. However, the area increase for pipeline and tiny vector instructions is about 3% in total.

Table 1 shows the frequency obtained in the post-place and the route simulation of MuCCRA-3, DRPA-base, DRPA-PL, and DRPA-TVI. The clock frequency of MuCCRA-3 is much lower than DRPA-base because of the island style network in the PE array. Pipelined PE improves the frequency of DRPA-PL and DRPA-TVI by 2.4 times as that of DRPA-base. A high clock frequency of 1.47GHz is achieved by using the pipelined structure and the point-to-point network.

4.3. Performance of application programs

Five application programs were used for evaluation. Alpha Blender, Sepia Filter and FIR Filter are simple image filters. DCT is discrete Cosine transfer used for JPEG encoder and SAD is sum of absolute difference used for template

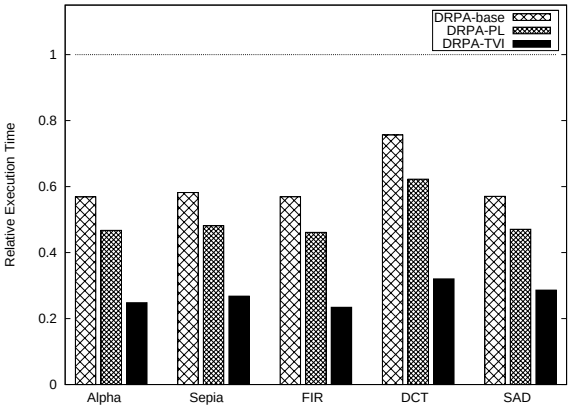


Fig. 4. Evaluation results: Relative execution time.

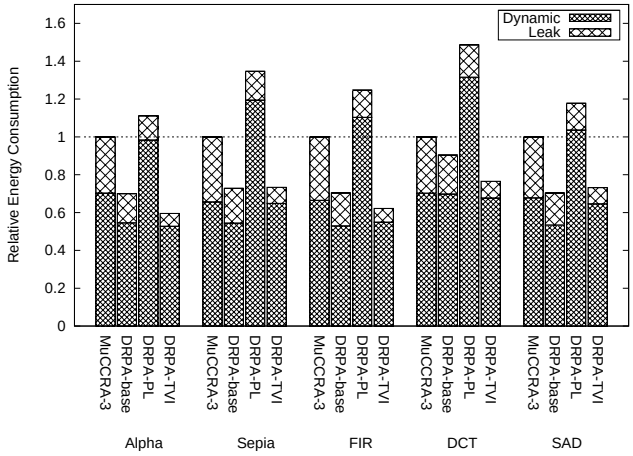


Fig. 5. Evaluation results: Relative energy consumption.

matching. Each program was described with an assembly tool called *mucasm2*, whose input file included an assembly code representing the operations of each PE and DMEM for all contexts. Routing between the allocated operations is performed automatically and configuration data to which RoMultiC bits are added for each generated context.

4.3.1. Performance

The relative execution times of applications to MuCCRA-3 are shown Fig. 4. Here, the loading time of configuration data to context memories from an external memory is not included. The data input/output can also be eliminated since it can be overlapped with the computation. The results are normalized to the execution time using DRPA-base. The vector length is set to two, as this is enough to remove all pipeline stalls.

Compared to DRPA-base, the performance improvement of DRPA-PL was only about 1.5 times due to frequent stalls caused by the data hazard. In contrast, DRPA-TVI improved the performance by the operational clock ratio of 2.4 times by eliminating the stalls with the tiny vector instructions.

4.3.2. Energy consumption

We calculated energy consumption from the results of power consumption and execution time. The relative energy consumption is shown Fig. 5. It is also normalized to MuCCRA-3.

DRPA-PL suffered from a severe increase of energy consumption due to the frequent stalls. In contrast, in DRPA-TVI there was almost no energy overhead since the execution time was reduced along with increasing the clock frequency.

5. CONCLUSION

In this paper, we proposed enhancing the operational speed of DRPAs by introducing a pipelined structure into each PE. In order to mitigate of the influence of data hazard between PEs, tiny vector instruction is introduced. With a single vector instruction, a small amount of data is continuously processed in the pipeline of the PE. Pipeline stalls are eliminated without increasing the number of hardware contexts and thus the amount of configuration data. DRPA with tiny vector instructions (DRPA-TVI) improved the performance by almost 2.4 times compared to a base DRPA with only a small increase of area and power consumption.

6. REFERENCES

[1] T. Toi et al., “Optimizing Time and Space Multiplexed Computation in a Dynamically Reconfigurable Processor,” in Prof. of FPT2013, 2013, pp. 106–111.

[2] J. Lee et al., “Real-time Ray Tracing on Coarse-grained Reconfigurable Processor,” in Prof. of FPT2013, 2013, pp. 192–197.

[3] H.Amano et.al, “MuCCRA Chips: Configurable Dynamically-Reconfigurable Processors,” *Proc. of ASSCC*, pp. 384–387, 2007.

[4] S. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe, and R. Taylor, “Piperench: a reconfigurable architecture and compiler,” *Proc. 26th Annual International symposium on Computer Architecture*, vol. 33, no. 4, pp. 70–77, 2000.

[5] R. M. Russell, “The CRAY-1 computer system,” *Commun. ACM*, vol. 21, no. 1, pp. 63–72, Jan. 1978.

[6] H. Kobayashi, R. Egawa, H. Takizawa, K. Okabe, A. Musa, T. Soga, and Y. Shimomura, “First experiences with NEC SX-9,” in *High Performance Computing on Vector Systems 2008*. Springer, 2009, pp. 3–11.