Performance and Power Analysis of Time-multiplexed Execution on Dynamically Reconfigurable Processor

Yohei Hasegawa[1], Shohei Abe[1], Shunsuke Kurotaki[1], Vu Manh Tuan[1],
Naohiro Katsura[1], Takuro Nakamura[2], Takashi Nishimura[2] and Hideharu Amano[2]

[1] Graduate School of Science and Technology, Keio University
[2] Faculty of Science and Technology, Keio University
3-14-1 Hiyoshi, Kouhoku-ku, Yokohama, 223-8522 Japan
drp@am.ics.keio.ac.jp

Abstract

Dynamically Reconfigurable Processor (DRP) developed by NEC Electronics is a coarse grain reconfigurable processor that selects a datapath called a context from the on-chip repository of sixteen circuit configurations at runtime. The time-multiplexed execution based on the multicontext functionality is expected to drastically improve area and power efficiency. To demonstrate the impact of the time-multiplexed execution, we have implemented several stream applications on DRP with various context sizes. Throughout the evaluation based on real application designs, we analyzed the impact of the time-multiplexed execution on performance and power dissipation quantitatively.

1. Introduction

In recent years, dynamically reconfigurable devices have attracted much attention because of their high-speed processing and flexible properties in embedded systems[11][3]. Recent dynamically reconfigurable processors, such as DRP[7], DAPDNA[4], XPP[8], and D-Fabrix[1], consist of coarse-grained processing elements (PEs) and distributed memory modules. They can dynamically change the PE operation and inter-PE connection according to the configuration instruction. Array of many PEs and memory modules provides high throughput in multimedia and network processings. Moreover, dynamic reconfigurability is expected to provide preferable area and power efficiency for the fraction of traditional programmable devices such as Field Programmable Gate Arrays (FPGAs).

Dynamically reconfigurable processors are also believed to be used as Intellectual Property (IP) cores for System-on-Chips (SoCs). In recent embedded products, the reduction of power dissipation become a significant challenge because of wide spread of battery-powered devices. The strong growth of wireless communications and portable applications is also a remarkable factor behind this trend. So, the power efficient architecture must be examined for future embedded systems.

The most remarkable characteristic of dynamically reconfigurable processors is the dynamic reconfigurability based on the multicontext functionality. The datapath realized on the physical hardware is called a *context*. Recent dynamically reconfigurable processors can change contexts often in one clock by the dynamic reconfiguration. The target application is divided into multiple contexts and only one context which is required at that point should be activated and executed. The multicontext functionality reduces the physical die size and improves power efficiency. We call this execution scheme based on the multicontext functionality the *time-multiplexed execution*.

However, there exist few works that clarify the impacts of the time-multiplexed execution on the power efficiency of the dynamically reconfigurable processor. Moreover, it's not apparent that how much degree of the time-multiplexing should be selected to optimize the power efficiency for the target application. The context size, which is defined by the amount of available computational resources, is a key factor that affects this issue directly. We should determine a particular context size so as to achieve required performance with optimal area and power efficiency for the target application.

The purpose of this work is to quantitatively clarify the impact of the time-multiplexed execution on performance and power dissipation. Throughout this study, NEC Electronics' Dynamically Reconfigurable Processor (DRP) is a target device. We have implemented several practical applications on DRP with various context sizes. So, we evaluate the power efficiency focused on real application designs.

1-4244-0054-6/06/$20.00

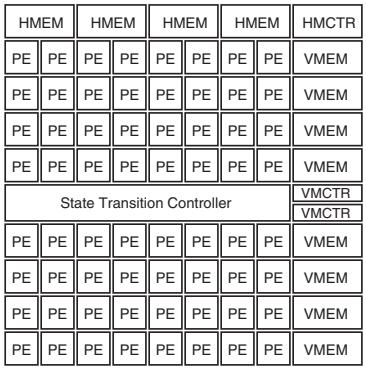


Figure 1. DRP Tile Architecture

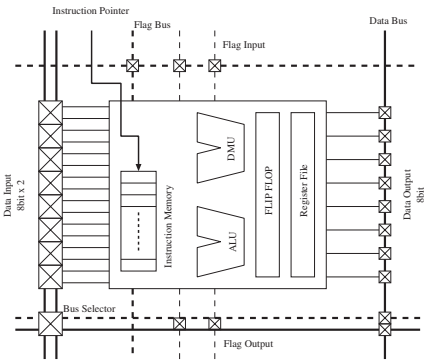


Figure 2. Processing Element Architecture

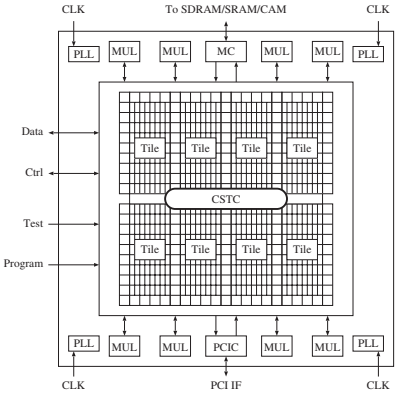


Figure 3. Prototype Chip: DRP-1

The rest of this paper is organized as follows. In Section 2, we describe the DRP architecture which is the target device throughout this paper. Section 3 presents the model of the time-multiplexed execution on DRP and its impact on performance and power dissipation. In Section 4, we describe evaluation results and discussions about power efficiency. Finally, we conclude this paper in Section 5.

2. Target Device: DRP

DRP is a coarse grain dynamically reconfigurable processor that was released by NEC Electronics in 2002 [7]. It carries an on-chip configuration data corresponding to multiple contexts, and it dynamically reschedules them to realize multiple functions with one chip.

Sixty-four of the most primitive 8-bit *processing elements (PEs)* are combined to form what is called a *Tile*, and the DRP Core consists of an arbitrary number of these Tiles. Figure 1 is a sketch of a Tile in DRP. The DRP architecture was designed with a focus on coupling with other cores on a SoC, and it has peripheral connections for such purposes. In each Tile, there are 64 PEs, one State Transition Controller (STC), eight 2-ported memories (VMEMs: Vertical MEMories), two VMEM Controllers (VMCTR), four 1-ported memories (HMEMs: Horizontal MEMories), and one HMEM Controller (HMCTR). The number of Tiles is expandable, horizontally and vertically.

The structure of a PE is shown in Figure 2. It has an 8-bit ALU, an 8-bit DMU (for shifts/masks), an 8-bit$\times$16-word register file (RFU), and an 8-bit flip-flop (FFU). These units are connected with each other by programmable wires specified by instruction data (configuration data), and their bit-widths range from 8 Bytes to 18 Bytes depending on the location. The PE has 16-depth instruction memories (e.g. 16 contexts) and supports multiple context operations. Its instruction pointer is delivered from the STC.

The STC is a simple sequencer in which any finite state machine (FSM) can be stored. The STC has 64 states, and each state is associated with the instruction pointer. The FSM of the STC operates synchronized with the internal clock, and generates the instruction pointer for each clock cycle according to the state. Also, the STC can receive event signals from PEs to branch conditionally. The maximum number of branches that can be specified from the PE is four.

As for the memory units, a Tile has eight 2-ported VMEMs and four 1-ported HMEMs on its vertical and horizontal sides respectively. The capacity of a VMEM is 8-bit$\times$256-word, and four VMEMs can be handled as a FIFO, using a VMCTR. HMEM is a single-ported memory and it has a larger capacity than the VMEM. It has 8-bit$\times$8K-word entries. Contents of these memories, flip-flops, and register files of the PE are shared with the datapath of all the contexts. The DRP Core, consisting of several Tiles, can change its contexts every cycle with the instruction pointer distributed from STCs. Also, each STC can run independently by programming different FSMs.

DRP-1, shown in Figure 3, is the prototype chip using the DRP Core with 4$\times$2 Tiles. It is fabricated with 0.15-μm CMOS processes. It consists of 8 Tiles, eight 32-bit multipliers, an external SRAM controller, a PCI interface, and 256-bit I/Os. The maximum operation frequency is 100-MHz. Although the DRP-1 is used as a stand-alone reconfigurable device, Tiles of DRP can be used as an IP core on SoCs with an embedded processor. In this case, the number of Tiles can be chosen so as to achieve the required performance with minimum area.

An integrated design environment, called Musketeer, is provided for the DRP-1. It includes a high level synthe-

sis tool, a design mapper for DRP, simulators, and a layout/viewer tool. Applications can be written in a C-based high level hardware description language, synthesized, and mapped directly onto the DRP-1.

3. Time-multiplexed Execution on DRP

3.1. The Basic Model

Dynamically reconfigurable processors are aimed at stream applications such as JPEG and MPEG for embedded systems. The target application or task is divided into multiple contexts and executed switching them at run-time. While the context switching is performed sequentially, a lot of PEs and distributed memory modules in a certain context are operated in parallel. So, the performance depends on the way of context scheduling.

Normally, to execute the target application, it takes a certain clock cycles because of the inherent sequentiality and parallelism. In particular, when a stream application is executed by DRP, we have the following typical processing flow.

1. Data is read out from distributed memory modules and/or registers,
2. required processing is performed with multiple PEs, then
3. the results are written back to distributed memory modules and/or registers.

Here, these contiguous processes are considered as a serial step in the algorithm. Usually the step is iteratively executed and every step requires at least a clock cycle, since results must be written into distributed memory modules and/or registers. After finishing all processes, the output data stream is flushed out from the chip.

The target application has a restriction of serial execution, and parallelism is defined in every serially executed steps. Throughout this work, we consider the number of required PEs as the parallelism for each step. Figure 4 shows the number of required PEs in each serial step when Discrete Cosine Transform (DCT) in JPEG encoder is executed on the DRP Core. In this diagram, PEs and VMEMs/HMEMs are assumed to be available without any restrictions. Note that, each step may be iteratively executed several times according to the algorithm.

The DCT algorithm mainly performs product-sum operations in row and column directions on an 8×8 image data. The behaviors of each step are as follows.

- Initialize distributed memory modules and registers and input data is stored (Step 0 - 6)
- Operate in the row direction (8-times loop of Step 7 -10)

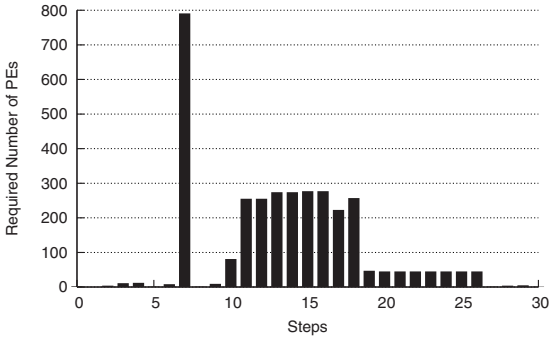


Figure 4. The Number of Required PEs for each step (DCT)

- Operate in the column direction (4-times loop of Step 11 - 18 and then Step 19 - 26)
- Output results (Step 27 - 30)

In this implementation, the 8×8 image data is stored into eight independent vectors so that they can be accessed in the row direction simultaneously. Once the data is ready, row-directional operations can be executed fully using nearly 800 PEs. In the DCT design, the multiplier factor is constant, and in such a case, the multiplication is transformed into shifts and additions automatically by the DRP compiler. Although the column direction access must be performed in the sequential manner, the pipelined accessing by multiple vectors can enhance the parallelism. Therefore, the sequential memory accesses in a column direction won't degrade the parallelism so much except the last iteration to store final results.

Formally, let S_∞ be the minimum number of required steps considering interations for executing a certain application with infinite PEs and VMEMs/HMEMs. In the case of the DCT shown in Figure 4, we have $S_\infty = 72$ ($4 \times 8 = 32$ for each direction and 8 for epilogue of column direction). Since S_∞ depends on the algorithm itself and the data structure, the DCT requires 72 clocks even with the infinite number of available PEs and VMEMs/HMEMs. Therefore, S_∞ also means the minimum number of execution clock cycles required for the target application on DRP.

Then, we consider the case that all steps are realized on a single context assuming that available PEs and VMEMs/HMEMs are boundless. This corresponds to the case without the time-multiplexed execution. Even in this case, the target application requires S_∞ steps, and then the execution time is represented as

$$T_\infty = C_\infty S_\infty$$

where the C_∞ is the maximum delay time in this case.

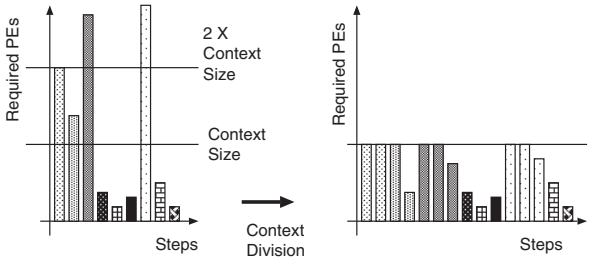


Figure 5. Context Division

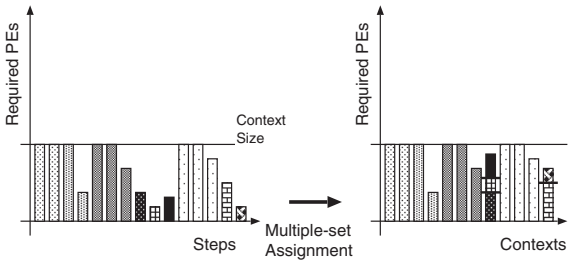


Figure 6. Multiple-Step-Allocation

In the time-multiplexed execution, it is a straightforward interpretation that a step operation is mapped to a context (PE array) and step transitions correspond to context switchings of DRP. However, since the number of available PEs in a context and the number of contexts are finite, the following techniques are actually needed to improve the resource utilization.

3.2. Context Scheduling Techniques

3.2.1 Context Division

In the time-multiplexed execution, each step is assigned to a context under the constraint of the number of available PEs per context (i.e. context size). If the number of required PEs per step exceeds this constraint, the step must be divided to multiple child steps as shown in Figure 5. In this paper, we call this technique *the context division*.

Formally, let PE_{size} be the number of available PEs per context and PE_i be the number of required PEs in step i $(i = 0, \cdots, S_\infty - 1)$. Given a certain context size, by the context division, the minimum number of required steps, S_{size}, is increased as shown in

$$S_{\text{size}} = \sum_{i=0}^{S_\infty - 1} \left\lceil \frac{PE_i}{PE_{\text{size}}} \right\rceil .$$

Since additional registers are inserted at division points for inter-step communications, the number of required PEs may be increased. In addition, for a certain context size, the maximum delay time C_{size} may be cut with the context division. Even without this division, the operational frequency will be increased in a small hardware context since the wire length is reduced. Considering the reduction of the maximum delay, the total time-multiplexed execution time T_{size} is given by

$$T_{\text{size}} = C_{\text{size}} S_{\text{size}}.$$

3.2.2 Multiple-Step-Allocation

As shown in Figure 4, there exist many steps which use few PEs because they handle the memory access only. Therefore, PE utilization is quite poor in this DCT implementation. Furthermore, the number of required contexts will also exceed the capacity of the DRP-1 (i.e. 16 contexts).

To overcome this problem, *the multiple-step-allocation* method packs multiple steps which have lots of unused PEs into one context keeping the context size fixed. Figure 6 shows the concept of the multiple-step-allocation method. STCs of the DRP Core determine the step to be activated according to state transitions of STCs. So, this technique can improve the PE utilization and reduce the number of contexts.

This technique surely contributes to improve PE utilization and reduce the number of required contexts, but never improve the execution time. The main reason is that S_{size} remains unchanged even with the multiple-step-allocation. In addition, since the whole datapath on a single context is spatially extended, the wiring delay usually grows longer. Furthermore, the additional multiplexers are necessary to share registers and distributed memory modules among multiple steps. This results in the increase of required PEs per context and the maximum delay time. Consequently, the execution time may be damaged by the multiple-step-allocation.

However, since the step including the maximum delay path commonly requires many PEs by itself, it is never selected as the target of the multiple-step-allocation. Therefore, it has little impact on the performance in the most cases. Note that, if all steps can be allocated into a single context using the multiple-step-allocation, it corresponds to the execution without time-multiplexing.

3.3. Impact of Time-multiplexing

In this section, we make qualitative discussions as to the impact of the time-multiplexed execution on performance and power dissipation.

The time-multiplexed execution based on the multicontext functionality provides high area and power efficiencies to embedded SoCs. This is derived from the property that the required context can be driven only when necessary. Thus, the target application can be executed consuming minimum hardware cost and power for the required per-

Table 1. Target Applications

Application Name	Abbr.	Max PEs	Ref.
Discrete Cosine Transform in JPEG encoder	DCT	259	[12]
Inverse Modified Discrete Cosine Transform in MP3 decoder	IMDCT	360	[12]
Fast Fourier Transform	FFT	101	[5]
Active Direction Pass Filter	ADPF	90	[5]
Viterbi Decoder	Viterbi	320	-
Advanced Encryption Standard on ECB mode	AES-ECB	448	[2]
Secure Hash Algorithm 1	SHA-1	61	[2]

formance. Here, the context size, which corresponds to the amount of available computational resources, is a crucial factor for the time-multiplexed execution. It is important to determine the optimal context size for the target application in order to optimize area and power efficiency.

If the context size is small, the physical hardware area is also small, but the context division results in increases of the required number of steps and contexts. Meanwhile, the operation frequency will be improved because the maximum delay path is also divided by the context division. If the number of PEs is small, the power dissipation of running PEs will be also small. Instead, the improvement of the operation frequency will increase the clock network power and frequent context switching will trigger the increase of the dynamic power dissipation.

On the other hand, the large size of context helps more parallel processings with more PEs and VMEMs/HMEMs. This will lead to achieve high throughput and reduce the required steps and contexts. Although the low operation frequency and low-frequent context switching reduces power dissipation, running PEs including ones in which any operations are not assigned will increase the power dissipation. Given an excessive context size, the inappropriate multiple-step-allocation may cause fatal losses of delay and resource cost because of increasing multiplexers as mentioned in Section 3.2.2.

When we select the context size for the target applications, we must consider above architecture trade-offs for the area and power efficiency. The following section describes quantitative evaluation results of power efficiency based on real application designs on the DRP-1.

4. Evaluation

In this section, we present quantitative evaluation results based on real application designs. We have implemented multiple stream applications on the DRP-1 with various context sizes and evaluated power efficiency for each design.

4.1. Target Applications

Many stream applications have been implemented on the DRP-1. We evaluate performance and power dissipation based on real application designs.

The target applications are listed in Table 1. The algorithms of these applications are described in Behavioral Design Language (BDL) which is the modified C-like language and compiled by the DRP compiler. Besides, "Max PEs", which is the maximum number of required PEs for each application at 8-Tile case, is also shown in Table 1. As mentioned in Section 3.1, since the parallelism is defined by the number of required PEs in our model, the value of "Max PEs" can be viewed as the maximum parallelism of the target application. From Table 1, it can be said that DCT, IMDCT, Viterbi, and AES-ECB have relatively higher degree of parallelism than FFT, ADPF, and SHA-1.

Here, in order to evaluate performance and power dissipation of the time-multiplexed execution, we designed and compiled each application with various context sizes. In this paper, the context size means the number of available PEs in one context physically. Since the basic building unit of DRP is the Tile, we can determine the context size of the DRP Core by the Tile. Each Tile has 64 PEs and a certain number of VMEMs/HMEMs.

The context scheduling including the context division and the multiple-step-allocation can be performed automatically or manually. In this evaluation, we optimized the context division for each context size manually because it largely affects to performance and power dissipation. In contrast, we leave the multiple-step-allocation to the DRP compiler unless the high-level optimization is required.

4.2. Evaluation Points

4.2.1 Performance

Performance of each application is measured by the execution time. In order to analyze the effect of the time-multiplexed execution to performance, we evaluate the relative execution time R_{size}. R_{size} shows the ratio of the execution time to T_∞ which is one without the time-multiplexed

execution. In the case without the time-multiplexed execution, we make an assumption that infinite PEs and VMEMs/HMEMs are available. So, R_{size} can be expressed as

$$R_{\text{size}} = \frac{T_{\text{size}}}{T_\infty} = \frac{C_{\text{size}} S_{\text{size}}}{C_\infty S_\infty}.$$

Since the number of required PEs in the case without the time-multiplexed execution is often over the number of available PEs on the DRP-1 (i.e. 512), it is impossible to layout the context on the DRP-1. So, there is no way to measure the real maximum delay C_∞. In this case, we use C_8 which is the maximum delay in 8-Tile case alternatively. Note that, since $C_\infty \geq C_8$ generally, R_{size} may be smaller than the case of using real C_∞.

4.2.2 Power and Energy Consumption

Power dissipation for each application is measured by the power profiler of the DRP-1. Since the profiler estimates the power dissipation for the DRP-1, some errors may be observed in this estimation compared with real power dissipation. Also, the profiler cannot include the power of peripheral circuits such as I/O because the power of the 8-Tile DRP Core of the DRP-1 chip is estimated. Moreover, since the profiler supports only 8-Tile case, we correct the estimated power for each context size.

Finally, we evaluate the required energy consumption E_{size} for each application with the context size. E_{size} is defined by the product of the power and the execution time. Let P_{size} be the power dissipation for each context size, so we have

$$E_{\text{size}} = P_{\text{size}} T_{\text{size}} = P_{\text{size}} C_{\text{size}} S_{\text{size}}.$$

Just like T_{size}, P_{size} and E_{size} are relatively evaluated by comparing to P_∞ and E_∞ which are power and energy consumption without time-multiplexing. However, P_∞ and E_∞ cannot be actually measured for the same reason of C_{size}. Therefore, we substitute them as P_8 and E_8 alternatively.

4.3. Results

In this section, we study how the context size affects performance and power dissipation. Figure 7 shows the relative execution time, power dissipation, and required energy consumption for each application with different context sizes. Note that each value is normalized to the case that the context size is 8-Tile.

Evaluation results of performance show that the execution time can be reduced by a larger context size. In particular, larger size of context can provide more performance improvement to highly parallel applications such as DCT and Viterbi. In contrast, for applications which have low parallelism and require small number of PEs such as FFT, ADPF, and SHA-1, it isn't expected to reduce execution clock cycles even with the larger context size. For example, in the case of SHA-1 and ADPF, there exists a limitation of performance improvement on 2 Tiles. This result also shows that the larger context size may cause performance degradation such as the case of IMDCT. This is because the inappropriate multiple-step-allocation causes a damage to the maximum delay.

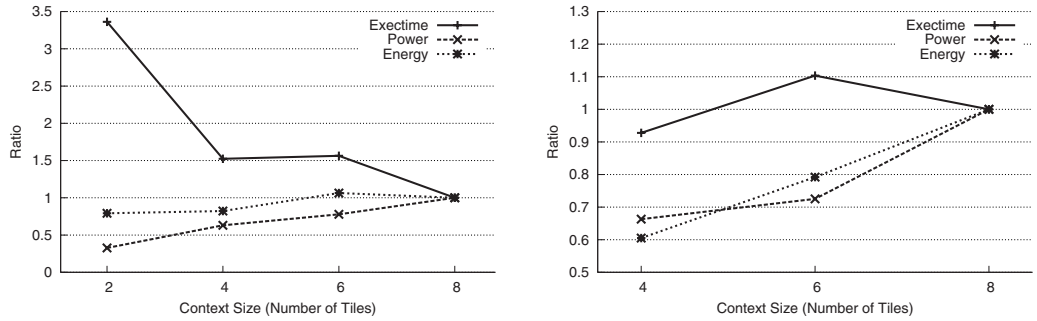
In contrast, the power dissipation increases if the context size becomes larger. This is because the larger size of context has a number of running PEs including ones in which any instructions are not assigned. Evaluation results show that when the context size keeps increasing, the power dissipation of the highly parallel application also increases. In the estimation of the power profiler, the power dissipation of DCT design with 2 Tiles is 113mW, and one with 8 Tiles is 345mW.

In the dynamically reconfigurable processor, it is afraid that the context switching facilities increase the additional dynamic power dissipation. In our evaluations, the context switching power of the DRP-1 turns out to be about 5% of the total power from statistical observations. Moreover, the power of fundamental parts such as clock network is dominant. So, the frequency of context switching isn't so influential for power dissipation. Therefore, when the context size keeps increasing, it is disadvantageous in terms of the power dissipation. And, the additional resources as costs of the context division and the multiple-step-allocation increase the power dissipation.

Although power dissipation increases with the larger context size, the reduction of the execution time can reduce the total energy consumption. In the case of DCT with 4 Tiles, since the execution time is substantially reduced compared with the 2-Tile design case, it is required the least energy. However, in most cases, larger context size is disadvantageous to the energy consumption. This is because the larger size of context could not bring out the sufficient parallelism of the target applications.

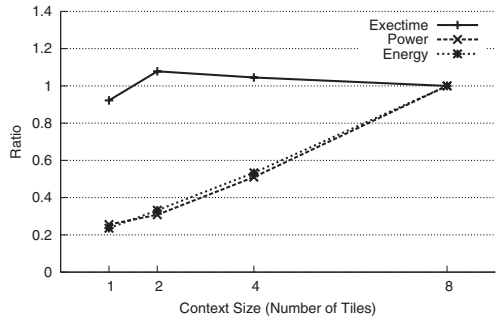
In particular, in the case of low-parallel applications such as FFT, ADPF, and SHA-1, the execution time cannot be improved even if the context size increases over the limitation of often 1 or 2 Tiles. For this reason, the power dissipation increases with enlarging the context size and the energy consumption also increases because of poor performance improvement. Hence, the small context size of about 2 Tiles is preferable for these applications with respect to the energy consumption.

Consequently, the remarkable insight is obtained from evaluation results. We summarizes that selecting the context size in which the cost-performance ratio is best results in the optical energy consumption for most applications.

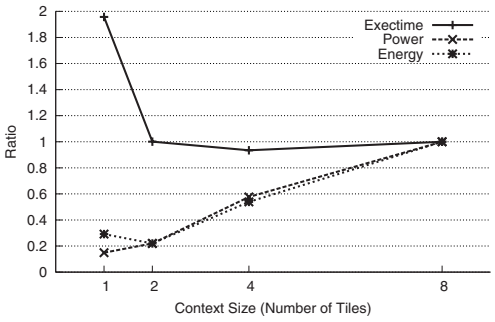


(a) DCT

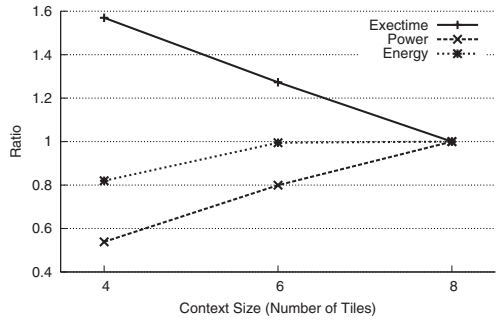
(b) IMDCT



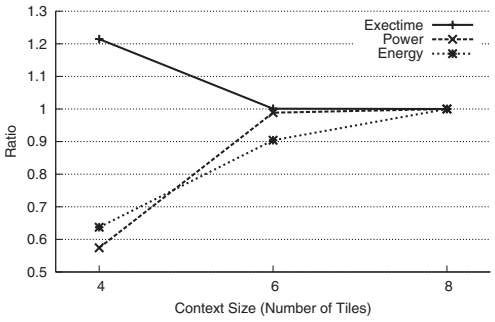
(c) FFT



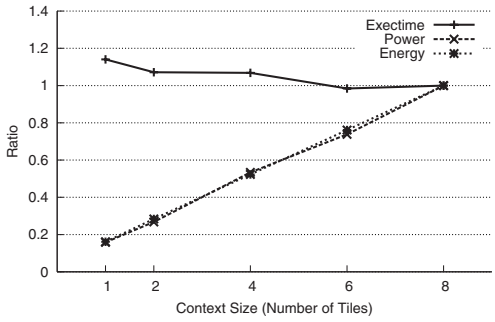
(d) ADPF



(e) Viterbi Decoder



(f) AES-ECB



(g) SHA-1

Figure 7. Performance, Power, and Energy Consumption v.s. Context Size

5. Conclusion and Future Work

This paper reports the real impact of the time-multiplexed execution with the dynamically reconfigurable processor focused on performance and power dissipation. We implemented multiple stream applications on the NEC Electronics' DRP-1. Each application is designed with various context sizes and evaluated for each context size.

At first, we measured the execution time of the target application for each context size. The applications with high-degree of parallelism have a possibility of performance improvements when the context size keeps increasing. In contrast, in the case of low-parallel applications, the execution clock cycles cannot be eliminated even if larger size of context is available. Instead, the multiple-step-allocation stretched the maximum delay and finally damaged the execution time.

From the point of view of power and energy consumption, we have the following fact. Although the context size keeps increasing and the power dissipation also increases, we can expect that the required energy consumption can decrease because of the significant reduction of the execution time. However, because of the poor performance improvement, our evaluation results show that the larger context size isn't preferable for most applications in terms of required energy consumption. In low-parallel applications, there exists no room of performance improvement even with the excessive size of context. Accordingly, the required energy is turned out to increase.

The parallelism and sequentiality of the target application must be observed preliminarily to optimize area and power efficiency. This observation will help us to determine the optimal context size of DRP Core as an IP core. In the case of DRP, we guess that this process can be easily performed by using the integrated design environment with general C-programming skills.

For more power-efficient architecture, we must investigate the power of dynamically reconfigurable processors in more detail. In particular, future work includes studying relations between power dissipation and frequency of the context switching. In addition, since many FPGA architecture evaluations for power efficiency have been already performed and future architectures are examined[10][9][6], the comparison of power efficiency with power-aware FPGAs such as Xilinx's Spartan devices[13] is required. Furthermore, we shall develop low-power application design techniques for dynamically reconfigurable processors.

Acknowledgment

Throughout this work, the DRP-1 Device and its design and synthesis tools were provided from NEC/NEC Electronics. The authors would like to show our gratitude to all members of the DRP development group at NEC Electronics and NEC Laboratories for their design tool support and considerable amount of technical advice. The part of simulations used in this project is supported by Mentor Graphics university program.

References

[1] Elixent. http://www.elixent.com/.

[2] Y. Hasegawa, S. Abe, H. Matsutani, K. Anjo, T. Awashima, and H. Amano. An Adaptive Cryptographic Accelerator for IPsec on Dynamically Reconfigurable Processor. In *Proceedings of IEEE International Conference on Field Programmable technology (FPT2005)*, pages 163–170, Dec. 2005.

[3] P. Heysters, G. Smit, and E. Molenkamp. A Flexible and Energy-Efficient Coarse-Grained Reconfigurable Architecture for Mobile Systems. *Journal of Supercomputing*, 26(3):283–308, Nov. 2003.

[4] IPFlex. http://www.ipflex.com/.

[5] S. Kurotaki, N. Suzuki, K. Nakadai, H. G. Okuno, and H. Amano. Implementation of Active Direction-Pass Filter on Dynamically Reconfigurable Processor. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-2005)*, pages 215–220, Aug. 2005.

[6] F. Li, D. Chen, L. He, and J. Cong. Architecture Evaluation for Power-Efficient FPGAs. In *Proceedings of ACM International Symposium on Field-Programmable Gate Arrays (FPGA2003)*, pages 175–184, Feb. 2003.

[7] M. Motomura. A Dynamically Reconfigurable Processor Architecture. *Microprocessor Forum*, Oct. 2002.

[8] PACT. http://www.pactcorp.com/.

[9] K. Poon, A. Yan, and S. Wilton. A Flexible Power Model for FPGAs. In *Proceedings of the International Conference on Field-Programmable Logic and Applications (FPL2002)*, pages 312–321, Sept. 2002.

[10] L. Shang, A. Kaviani, and K. Bathala. Dynamic Power Consumption in Virtex-II FPGA Family. In *Proceedings of ACM International Symposium on Field-Programmable Gate Arrays (FPGA2002)*, pages 157–164, Feb. 2002.

[11] G. J. M. Smit, P. J. M. Havinga, L. T. Smit, and P. M. Heysters. Dynamic Reconfiguration in Mobile Systems. In *Proceedings of International Conference on Field Programmable Logic and Application (FPL2002)*, pages 162–170, Aug. 2002.

[12] M. Suzuki, Y. Hasegawa, Y. Yamada, N. Kaneko, K. Deguchi, H. Amano, K. Anjo, M. Motomura, K. Wakabayashi, T. Toi, and T. Awashima. Stream Applications on the Dynamically Reconfigurable Processor. In *Proceedings of International Conference on Field Programmable Technology (FPT2004)*, pages 137–144, Dec. 2004.

[13] Xilinx. http://www.xilinx.com/.