# A Field-Customizable and Runtime-Adaptable Microarchitecture

Toshinori Sato

PRESTO
Japan Science and Technology Agency
toshinori.sato@computer.org

Daisuke Morishita

Department of Artificial Intelligence
Kyushu Institute of Technology
dsuke@mickey.ai.kyutech.ac.jp

**Abstract— This paper introduces our on-going project, titled *Study on Hardware/Software Adaptable Microprocessors* [10]. The goal of the project is to investigate dynamically adaptable microprocessors that utilize dynamic profile information. In this architecture, microprocessor hardware itself is not reconfigured. Instead, software, which consists of law control signals, is dynamically optimized, resulting in virtual reconfiguration of the hardware. This adaptability will provide us low-power and high-performance embedded processors.**

## I. Introduction

We are investigating a processor architecture, which is customized for a specific application when it is running[10]. While there are a lot of studies on field programmable computing[2, 7, 22], we have come up with starting yet another study on adaptable processor architectures, when we became aware of the following problem. To benefit from field programmable architectures, we have to perform hardware/software partitioning to accelerate hot spots in program execution. This is very tedious work for software engineers who expect performance benefit from dedicated hardware devices. Solving this problem is a key issue that field programmable architectures have big commercial success, and this is the goal of this study. To attain the goal, we revisit two historical architectures, dynamic microprogramming[15] and dataflow machines[4], and combine them with dynamic software optimization techniques[21].

## II. Customizable/Adaptable Microarchitecture

*Why reconfigurability is required ?*

One of the most important constraints in modern microprocessor design is power consumption. Because there is a tradeoff between performance and power, each microprocessor should be customized to a specific application in order to achieve both high-performance and low-power. In addition, it is better that the microprocessor is adaptable to each phase in the execution of the application, because every phase requires different hardware resources in microprocessor. On the other hand, this field customizability is desirable for the recent trend of shortening time-to-market requirement.

*What is reconfigured ?*

Our major target is embedded processors for intelligent mobile devices. One of the dominant applications is digital signal processing, which is computing-intensive. Hence, datapath should be customized to each application. On the other hand, cache memories are a major energy consumer of the entire processor. Thus, we also should consider adaptable cache memories for power reduction.

*How is a desirable configuration found ?*

To accurately recognize the characteristics of a specific application, profiling is a powerful tool. We utilize both static and dynamic profile information to determine optimal configuration.

*When is the reconfiguration done ?*

As mentioned above, dynamic hardware/software partitioning for reconfiguration is a key issue for field programmable devices. Thus, the reconfiguration should be done during an application program is running on the device. It is also possible to adapt the device to environment or to input data set using runtime information. This does not deny static reconfiguration. If static profile information is available, the compile-time customization results in better configuration. Hence,

we also customize a processor to a specific application when the binary code of the application is generated. This is easier than dynamic reconfiguration and is possible because resource requirements is determined when the algorithm for the application is fixed.

*Where is the configuration determined ?*

If large power consumption and computing power are not required for the optimization of hardware configuration, it is done on the running microprocessor that is the object for the reconfiguration. This can utilize on-time information. We expect this is possible since the scope of dynamic optimization is very limited. On the other hand, if large power consumption and computing power are required, the optimization is performed outside the processor, such as remote servers providing optimization services. An example of the remote optimization is that for mobile devices with wireless communication facility, such as future smart cell phones. A phone sends condensed dynamic profile information to the remote server, where optimal hardware configuration for the phone is explored.

## III. Reconfiguration via Software Optimization

Figure 1 shows the block diagram of the microarchitecture under consideration. It mainly consists of an customizable ALU array and adaptable caches. The ALU array resembles NEC's DRP[13] and UT-Austin's Grid[14], and dataflow machines[4] are mapped on the array. The difference from the related studies is in its configuration strategy. DRP uses conventional FPGA-like configuration data, and the Grid processor statically determines dataflow mapping on the ALUs. In contrast, we dynamically construct and adapt dataflow using micro-instruction memory in each tile, which they decide the behavior of the ALU and switch in the tile. The adaptation used to utilize the virtual machine[20] or the dedicated hardware[24]. Instead, it is done by replacing the micro-instructions stored in the micro-instruction memory. Each instruction is divided into several micro-instructions and then they are spread into the ALU array. This resembles dynamic programming for architecture tuning[1, 5, 9, 12, 15]. However, we would rather rely on dynamic optimization techniques[21] for the adaptation. Based on runtime profile information, instructions and thus micro-instructions are dynamically optimized, and thus the hardware is virtually reconfigured.

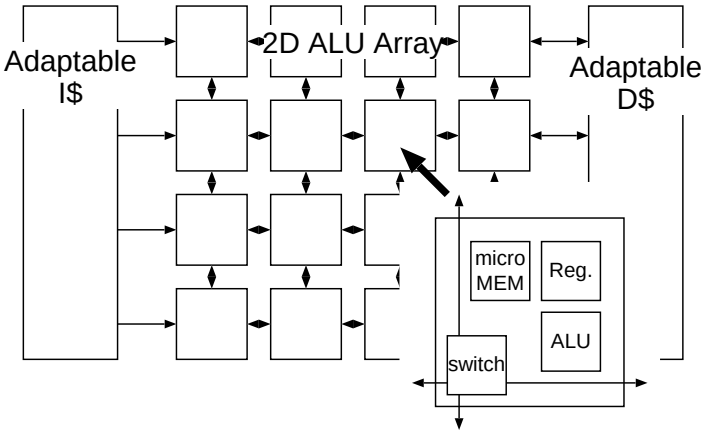


FIG. 1 Block Diagram

One of the dynamic optimization techniques is collapsing ALU array. This collapsing combines the staggered ALU[8] and the collapsed ALU[23], Furthermore, this utilizes the constructive timing violation technique[19], and enables to execute dependent operations in one cycle by dynamically identifying small delay operations. This resembles the technique proposed by Brooks[3], however, it exploit the large number of ALUs and improve their utilization.

On the adaptable memorie structures, we proposed techniques for reducing leakage current of caches[16, 17, 18] and the one for reducing power consumed in branch target buffers[18].

## IV. Improving ALU Utilization

This section presents preliminary results. Because the proposed microarchitecture relies on large ALU array, in this paper we first evaluate how efficiently a large number of ALUs can be utilized. And then, the impact of ALU collapsing in processor performance is evaluated.

We use MASE simulation infrastructure[11] for this evaluation. Because we are interested in the potential of a large number of ALUs, some microarchitectural features are idealized in this study. We use perfect caches, perfect branch prediction, and perfect memory disambiguation. We vary instruction window size between 64 and 1024. The number of ALUs are varied between 16 and 1024, and there is no limitation in the network connecting the ALUs.

TABLE I
BENCHMARK PROGRAMS

| program | input set |
|---|---|
| FFT | Fast Fourier Transform |
| Toast | GSM encode |
| CRC | 32-bit Cyclic Redundancy Check |

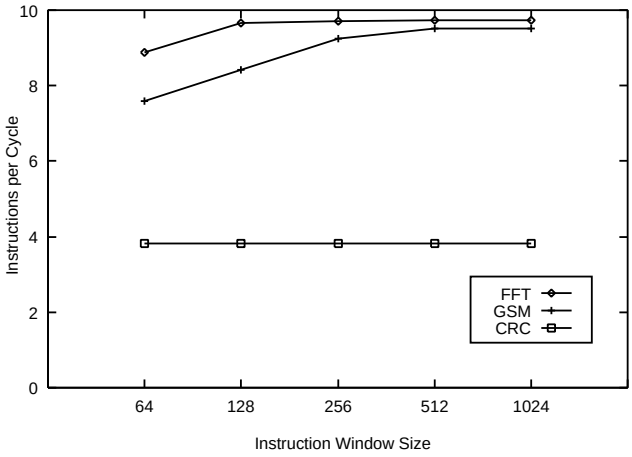


FIG. 2 EFFECT OF INSTRUCTION WINDOW SIZE ON PERFORMANCE

Three benchmark programs from MiBench[6] is used for this study. It is developed for use in the context of embedded, multimedia, and communications applications. It contains image processing, communications, and DSP applications. We use original input files provided by University of Michigan. Table 2 lists the benchmarks we used. All programs are compiled by Compaq C V6.1-120 on Digital UNIX V4.0G (Rev.1530) with the optimization options specified by University of Michigan. Each program is executed to completion.

First, we evaluate the effect of instruction window size. In this evaluation, the number of ALU is fixed to be 1024. Simulation results are shown in Figure 2. The horizontal line indicates instruction window size, and the vertical line indicates instructions per cycle (IPC) as the metric for evaluating processor performance. We can easily find that instruction window size over 128 has little impact on performance even when there are a large number of ALUs.

Next, we evaluate the effect of the number of ALUs. In this evaluation, instruction window size is fixed to be 1024. Simulation results are shown in Figure 3. The

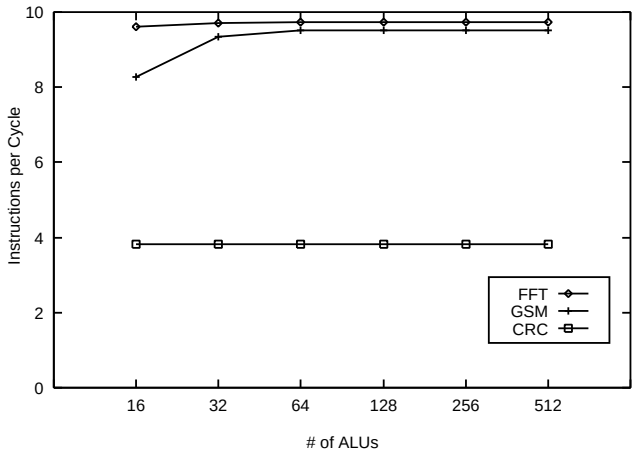


FIG. 3 EFFECT OF THE NUMBER OF ALUS ON PERFORMANCE

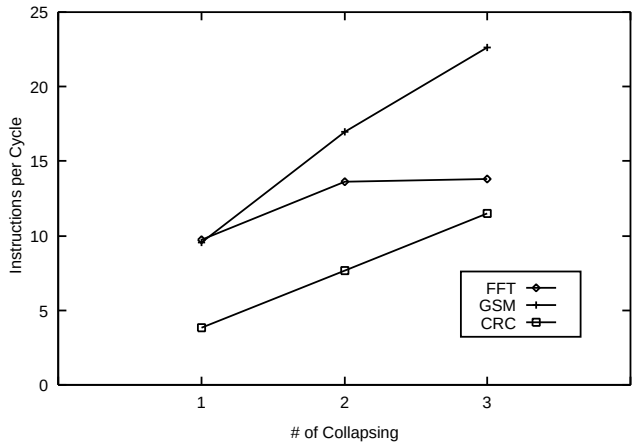


FIG. 4 EFFECT OF COLLAPSING ON PERFORMANCE

horizontal line indicates the number of ALUs, and the vertical line indicates IPC. We can also find that the number of ALUs over 32 has almost no impact on performance when instruction window size is enough large.

Last, we evaluate the effect of the number of ALU collapsing per cycle. In this evaluation, instruction window size is fixed to be 1024, and the number of ALU is also fixed to be 1024. Simulation results are shown in Figure 4. The horizontal line indicates the number of collapsing, and the vertical line indicates IPC. Different from the results presented above, ALU collapsing has significant contribution to processor performance. We can find almost linear effect on performance. The results encourage us to investigate dynamic optimization on collapsing ALUs based on exploiting small operand bitwidth.

## V. Summary

In this paper, we introduced the project, titled *Study on Hardware/Software Adaptable Microprocessors*. A major target is embedded processors for intelligent mobile systems. In such devices, both high-performance and low-power are strongly required. We believe the adaptability is inevitable for these requirements. Thus the goal of the project is to investigate dynamically adaptable microprocessors that utilize dynamic profile information. In this architecture, microprocessor hardware itself is not reconfigured. Instead, software, which consists of law control signals, is dynamically optimized, resulting in virtual reconfiguration of the hardware.

Our future study includes further investigating the ALU array, the remote optimization service, profiling methodology, and condense profile information. We also have to consider dynamic optimization method of instructions.

## References


[1] A.M.Abd-Alla, L.H.Moffett, "On-line architecture tuning using microcode," 3rd International Symposium on Computer Architecture, 1976.

[2] F.Barat, R.Lauwereins, "Reconfigurable instruction set processors: a survey," 11th International Workshop on Rapid System Prototyping, 2000.

[3] D. Brooks, M. Martonosi, "Dynamically exploiting narrow width operands to improve processor power and performance," 5th International Symposium on High Performance Computer Architecture, 1999.

[4] J.B.Dennis, D.P.Misunas, "A preliminary architecture for a basic dataflow processor," 2nd International Symposium on Computer Architecture, 1974.

[5] K.A.El-Ayat, J.A.Howard, "Algorithms for a self-tuning microprogrammed computer," 10th Workshop on Microprogramming, 1977.

[6] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, R. B. Brown, "MiBench: a free, commercially pepresentative embedded benchmark suite," Workshop on Workload Characterization, 2001.

[7] R.Hartenstein, "A decade of reconfigurable computing: a visionary retrospective," Design, Automation and Test in Europe Conference, 2001.

[8] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, P. Roussel, "The microarchitecture of the Pentium 4 processor," Intel Technical Journal, issue Q1, 2001.

[9] C.Ishikawa, K.Sakamura, M.Maekawa, "Adaptation and personalization of VLSI-based computer" architecture, 14th Workshop on Microprogramming, 1981.

[10] JST, `http://www.jst.go.jp/pr/report/report188/`, 2001 (in Japanese).

[11] E. Larson, S. Chatterjee, T. Austin, "MASE: A novel infrastructure for detailed microarchitectural modeling," International Symposium on Performance Analysis of Systems and Software, 2001.

[12] E.Luque, J.Sorribes, A.Ripoll, "Tuning architecture at runtime," 20th Workshop on Microprogramming, 1987.

[13] M.Motomura, "A dynamically reconfigurable processor architecture," Microprocessor Forum, 2002.

[14] R. Nagarajan, K. Sankaralingam, D. Burger, S. W. Keckler, "A design space evaluation of Grid processor architecture," 34th International Symposium on Microarchitecture, 2001.

[15] T.G.Rauscher, A.K.Agrawala, "Dynamic problem-oriented redefinition of computer architecture via microprogramming," IEEE Transactions on Computers, Vol.C-27, No.11, 1978.

[16] A.Sakanaka, T.Sato, "Reducing static energy of cache memories via prediction-table-less way prediction," 13th International Workshop on Power And Timing Modeling, Optimization and Simulation, 2003.

[17] A.Sakanaka, T.Sato, "A leakage-energy-reduction technique for high-associativity caches in embedded systems," 4th Workshop on Memory Access Decoupled Architectures and Related Issues, 2003.

[18] H.Sato, T.Sato, "A static and dynamic energy reduction technique for I-cache and BTB in embedded processors," Asia and South Pacific Design Automation Conference, 2004.

[19] T.Sato, I.Arita, "Constructive timing violation for improving energy efficiency," in L.Benini, M.Kandemir, J.Ramanujam "Compilers and operating systems for low power," Kluwer Academic Publishers, 2003.

[20] J.E.Smith, "Instruction-level distributed processing," IEEE Computer, Vol.35, No.4, 2001.

[21] M.D.Smith, "Overcoming the challenges to feedback-directed optimization," Workshop on Dynamic and Adaptive Compilation and Optimization, 2000.

[22] R.Tessier, W.Burleson, "Reconfigurable computing for digital signal processing: a survey," Journal of VLSI Signal Processing, Vol.28, 2001.

[23] S. Vassiliadis, J. Phillips, "Interlock collapsing ALUs," IEEE Transactions on Computers, Vol.42, No.7, 1993.

[24] B.Xu, D.H.Albonesi, "Runtime reconfiguration techniques for efficient general-purpose computation," IEEE Design & Test of Computers, Vol.17, No.1, 2000.