SysteMorph Prototyping on DAP/DNA

Yoshida, Makoto
Fukuoka Industry, Science & Technology Foundation FLEETS (Fukuoka Laboratory for Emerging & Enabling Technology of SOC)

Soga, Takeshi
Fukuoka Industry, Science & Technology Foundation FLEETS (Fukuoka Laboratory for Emerging & Enabling Technology of SOC)

Yoshimatsu, Norifumi
Fukuoka Industry, Science & Technology Foundation FLEETS (Fukuoka Laboratory for Emerging & Enabling Technology of SOC)

Murakami, Kazuaki
Department for Informatics, Graduate School of Information Science and Electrical Engineering Kyushu University

出版情報：Proceedings of 2004 IEEE Asia-Pacific Conference on Advanced System Integrated Circuits || || p420-423, pp.420-423, 2004-08. IEEE Asia-Pacific Conference on Advanced System Integrated Circuits
バージョン：
権利関係：

SysteMorph Prototyping on DAP/DNA

Makoto YOSHIDA † Takeshi SOGA † Norifumi YOSHIMATSU † and Kazuaki MURAKAMI † †
† Fukuoka Industry, Science & Technology Foundation
FLEETS (Fukuoka Laboratory for Emerging & Enabling Technology of SOC)
FS501,KASTEC 6-1 Kasuga-koen, Kasuga, Fukuoka 816-8580, Japan
† † Department for Informatics, Graduate School of Information Science and Electrical Engineering
Kyushu University
6-1 Kasuga-koen, Kasuga, Fukuoka 816-8580, Japan

***Abstract*-Dynamic optimization for software and hardware is gaining attention of computer system researchers. We are researching Systemorph technology. SysteMorph is a dynamic optimization technology proposed by Kyushu University. In this Paper, we discuss SysteMorph prototyping using DAP/DNA to evaluate Systemorph technology.**

***Index Terms*-dynamic optimization, reconfigurable computing, online profiling and online synthesis.**

I. INTRODUCTION

Dynamic optimization system is gaining attention of computer system researchers. As semiconductor technology advances, the scale of circuit and the cost of system LSI design are increasing drastically. Especially, in the high performance application specific system LSI, the balance of design cost, performance and power consumption is becoming challenging.

We are investigating the dynamic optimization technology called SysteMorph[1][2]. SysteMorph is feedback-directed dynamic software/ISA (instruction set architecture)/hardware co-optimization technology mainly realized in a processor base system. We are doing research to improve whole system performance using the SysteMorph technology. This dynamic optimization technology enables to optimize the system in accordance with changing system's behavior of actual situation. We are researching implementation of SysteMorph.

SysteMorph enables dynamic optimization using online profiler such as Dynamo[4]. Online profiler can collect information and predict of program behavior. It allows optimizer to optimize portions those are frequently executed in the program until the end of program. However, online profiler is executed while user program running on the same processor. We need to consider not only accuracy of prediction but also the overhead of execution of the profiler and cost of implementation.

In this paper, we introduce our online profiling method and online synthesis to realize the SysteMorph technology on the system which has a processor and a reconfigurable co-processor. Now, the SysteMorph concept is verified by making DAP/DNA-HP[3] of IPflex into a prototyping platform.

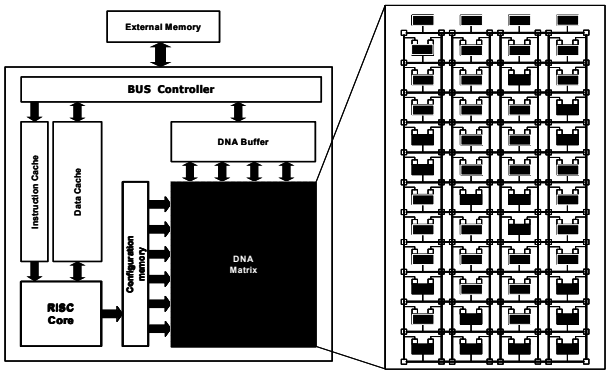


Fig. 1. DAP/DNA-HP

DAP/DNA-HP is a dynamic reconfigurable processor which consists of a RISC processor called DAP (Digital Application Processor) and a reconfigurable co-processor called DNA (Distributed Network Architecture) (Fig. 1).

We explains the technique of implementing a SysteMorph concept using the DAP/DNA.

II. SYSTEMORPH

SysteMorph is a Feedback-Directed Dynamic Software/ISA (Instruction Set Architecture)/Hardware Co-optimization technology mainly realized in a processor base system (Fig. 2). SysteMorph is a technology for optimizing performance, power consumption, and consumption energy. It is defined as SysteMorph being the technology for changing software, ISA and the mode/composition of hardware based on the profile information on the target application program, while performing the application concerned, and realizing the above-mentioned optimization.

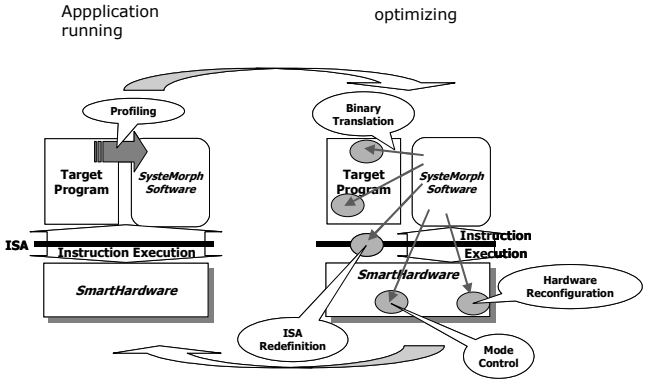


Fig. 2. SysteMorph

II. SYSTEMORPH ON DAP/DNA

A. SysteMorph implementation using DAP/DNA-HP

This chapter explains how to implement SysteMorph using DAP/DNA-HP (Fig. 3).

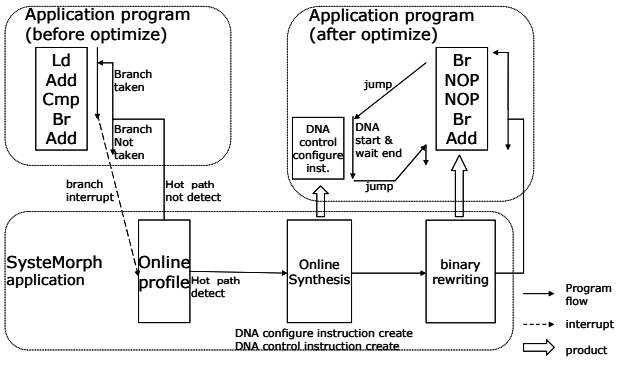


Fig. 3. SysteMorph implementing to DAP/DNA-HP

SysteMorph uses three-phase approach. (Fig. 3. lower berth).

1. Online profiling
2. Online synthesis
3. Binary rewriting

In the first step, online profiler on DAP collects execution frequency information for the program path. Our online profiler predicts execution frequency path (Hot Path) of a program. Online profiler accumulates branch history. Online profiler considers past frequency executed path of hot path which can be executed frequently in the future. In this paper, we predicts only loop hot path. The online profiler is accuracy in the point of considering each branch direction probability. We introduce online profiler we called BH (Branch History hot path prediction) method in the next section. When Hot Path is detected, online synthesis is performed.

In online synthesis, the Hot Path presumed by the profiler is analyzed, and mapping to DNA is performed. Online synthesis generates the instructions for DNA configuration information and DNA control execution as an output.

Binary rewriting is performed in order to execute the instructions generated by online synthesis. The head instruction of Hot Path rewrites to a jump instruction to the head address of the new generated instructions. Then, SysteMorph application returns from interruption to application program.

B. Online Profiling for DAP/DNA-HP

This section explains the needs of online profiling technique.

The online profiling is a technique to collect the information of processors' execution behavior online, while the offline profiling can collect whole information of program. So, the online profiling is different from offline profiling, the way to get the hint of optimization of the processors' execution behavior. The online profiling should satisfy the following properties:

1. low overhead
2. low cost
3. accuracy of prediction

Since the online profiler collects information while application programs are run, it should not have big impact for a whole system performance. So, we have to decrease the profiler overhead as possible as we can. And, the profiler logic should be put into a system with low cost. In short, we need decrease the execution time of profiler, and also we need to decrease the size of profiler's hardware circuit and its power consumption.

Online Profiling flow

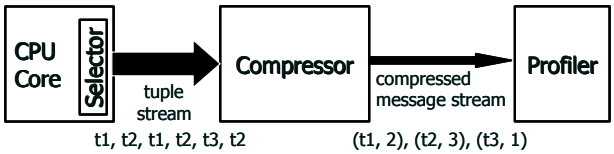


Fig. 4. Online profiler flow

The profiler consists of a selector, a compressor and a software profiler. The selector chooses some machine instructions that CPU Core executes. And, the selector generates tuple stream. Then, the compressor generates compressed message stream. At the end, the software profiler analyzes the compressed message stream and it gives a prediction where is executed frequently in future. The profiler's execution time overhead is minimized by using the hybrid composition of the hardware selector, the hardware compressor and the software profiler.

BH method algorithm

While a number of techniques have been proposed for collecting path profiles [4][5], there are still only a few methods to apply with low overhead execution time and small hardware size. NET [4] is low overhead method. However, NET uses executed path only when it found hot path tail. So, NET can predict wrong hot path by first executed path found hot path tail. We propose our online profiling algorithm which we call BH (Branch History hot path prediction) method.

The difference of BH method and NET method is the way to predict hot path. BH method can predict hot path more accurate than NET method. And also, BH method has lower overhead in terms of execution time than the other efficient profiling methods [5].

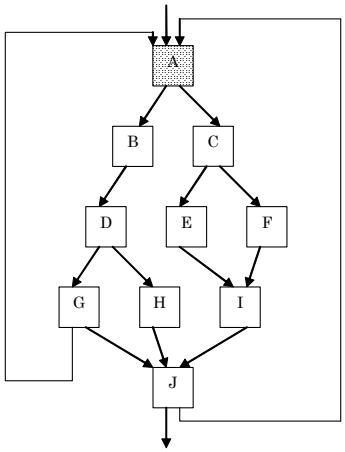
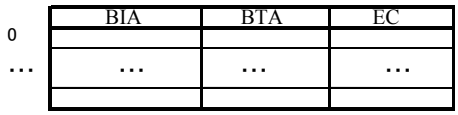


Fig. 5. Example of program path



BIA: Branch Instruction Address
BTA: Branch Target Address
EC: Executed Count

Fig. 6. Branch history table

Consider a loop path in figure 5 (A-B-D-G-A, A-C-F-I-J, etc.). The blocks (A to J) are basic blocks those blocks having a branch instruction at the bottom of the each block.
Our solution consists of four steps:

Step1: Selection of executed instructions: The profiler selects only branch instructions which CPU executes. And, it also gets the branch instructions' address and branch target address. Tuple stream consists of the pair of branch instruction address and branch target address.

Step2: Compression of tuple streams: Tuple stream is generated every time CPU executes branch instruction. Since branch history is analyzed to predict the hot path later, the compressor compresses the tuple stream by

merging the same entry (BIA, BTA) into the one entry (BIA, BTA, Count). Finally, the compressor generates compressed message stream. The compressed message stream consists of branch instruction address, branch target address and the count executed in the same BIA and BTA.

Step3: Prediction of hot path heads: The profiler analyzes the accumulated message streams. And it selects the message streams which have backward branch instruction and its count number is larger than some threshold. In figure 5, backward branch instructions are in block G and J. We consider this instruction address is head of hot path.

Step4: Prediction of hot paths: The hot path predictor predicts hot path by using hot path head and branch history table, accumulated by the message stream. At first, we searches the entry which hot path head addresses equals BIA from branch table. In the case the hot path head addresses are found more than two, one of the entries is chosen which count number is larger than the other one. At the second, selected entry's BTA address is considered as part of hot path. At the third, we search the branch instruction address (top of basic blocks) which equals the BTA from object code. At the fourth, we search next branch instruction from object code. At the fifth, we search BIA entry which equals its branch address of the object code from branch history table. We can get hot path if we repeat from second to fifth process until BTA address equals hot path head address.

The Step4 (Prediction of hot path) is considered that profiling overhead is increased because the traversing (searching from object code) of the object code needs. So, another scheme is introduced which does not require traversing (only searching from branch history table).

In this scheme the extended branch history table is introduced (Fig. 7.).

	BSA	BIA	BTA	EC
0				
…	…	…	…	…

BSA: Basic block start instruction address

Fig. 7. The extended branch history table

Here, basic block start instruction address (BSA) is branch target address. We call this branch history table extended branch history table.

In the selection step, branch target addresses equals the basic block start instruction addresses (BSA). In this step, the BSA is added to the branch history table with BIA, BTA and Count. Then the message streams are compressed. The step3 method is fully same.

In the step4, hot paths can be predicted by using only extended branch history table instead of by traversing object code. Because, all the program path information can be got by basic block head address. When we decide a frequent executed branch direction, its BTA equals BSA. We can simply get BSA with the buffer of the previous executed BTA. So, if we can find just BSA in the table entry, we can search the next BIA and BTA without traversing the object files. We can predict hot path with only searching table.

C. *Online synthesis for DAP/DNA-HP*

Figure 8 is the Whole flow of online synthesis for DAP/DNA

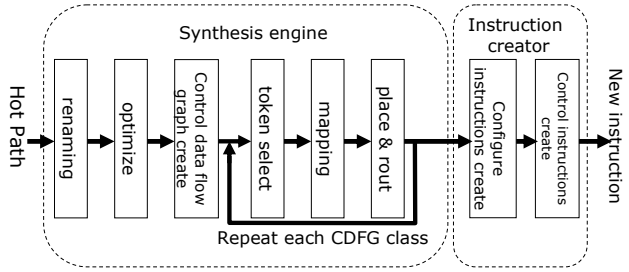


Fig. 8. The whole flow of online synthesis for DAP/DNA.

Renaming: The "renaming" stage changes "Hot Path" into SSA (static single assignment) form.

Optimize: The "optimize" stage deletes an unnecessary and redundant instruction.

Control-data flow graph (CDFG) creates: The "control-data flow graph create" stage derives the dependency during each instruction.

Token select: The "token select" stage chooses which DNA execution unit output signal is used for the signal for starting a DNA execution unit.

Technology mapping: The "technology mapping" stage changes an instruction to be expressed as a DNA function.

Place & Rout: The "place & Rout" stage does placement and routing the circuit mapped by DNA function on DNA.

Configure instructions create: The "configure instructions create" stage creates DAP machine instructions for writing DNA configuration memory from mapping, placement and routing information.

Control instructions create: The "control instructions create" stage creates DAP machine instructions for register stack, DNA start/stop, DNA end check, etc.

Token select

DAP/DNA uses the "token" as a trigger to change the output flip-flop of execution units. The token accompanies the data input which has two lines in an execution unit, chooses one of tokens at writing configuration to configuration memory. Output data changes, only when an input token is effective, and it holds a former state except case. An output token becomes effective only when an input token is effective, otherwise it is in an invalid state. Therefore, in order for an execution unit to take out the right output, it is necessary to choose the input which effective data reaches late as a token input. Moreover, when there is a dependency of a control, it is necessary to replace the token of a control with one token of the data input, and to choose the token as an input token with an execution unit.

The token is selected with the following rules.

Timing match: Input token is selected with a token out of the latest in the output data included in the last CDFG class.

Memory load: Memory load is the latest output token in the same CDFG class.

Loop connection: The input token of the first CDFG class is selected the token to start DNA and the output token of the last CDFG class.

In order to decide the input token of each class, since it is necessary to know exact cycle time that each output token belonging to the last class is generated, "token select" is performed after the completion of "Place & Rout" of the last class. However, at the time of token selection of the first class, placement and routing of the last class have not finished. At the time of token selection of the first class, all the output token of the last class are outputted in shortest time except memory load data. The cycle time is adjusted in order to

all the output tokens from the last class can occur simultaneously at the time of the last class placement.

DNA control flow

DNA is controlled by the instructions built at "configure instructions create" and "control instructions create" stage (Fig. 9).

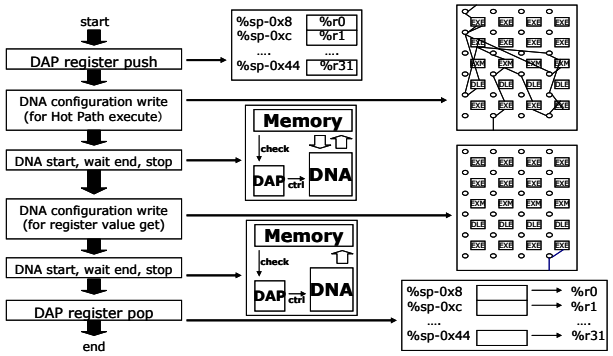


Fig. 9. DNA control flow

(1) In order to protect the register value of a user program, the register value pushes to stack area.

(2) Configuration data synthesized from Hot Path is written in configuration memory of DNA.

(3) Starting, checking end and stopping of DNA are performed.

(4) In order to return processing to DAP from DNA, configuration data for outputting the last register value which remains inside DNA to stack area is written in configuration memory.

(5) Starting, checking end and stopping of DNA are performed.

(6) Pop the register value from stack area.

Hot Path is the loop portion of a program. Therefore, the information about an initial-setting value is not contained, or an initial value may change each time at Hot Path execution. As a result, the mechanism which can write register value at the time of a Hot Path start in a DNA configuration writing memory is needed for the "control instructions create" stage.

In order to solve this problem, (a) the register value pushed to stack area is taken out. (b) Write the taken-out value in configuration memory. It is necessary to create instructions which realizes operation of the above (a) and (b) by "configure instruction create" stage.

IV. Experiment

SysteMorph application has been developed using the technique of having written in Chapter 3. On DAP/DNA-HP it succeeded in detecting Hot Path about dozens machine instructions and making it execute on DNA

IV. Conclusions

In this paper, the SysteMorph technical implementing technique with DAP/DNA-HP of IPflex has been introduced.

About online profiler, we proposed a branch history hot path prediction method.

Our approach can achieve accurate hot path prediction while it requires low profiler's execution time overhead.

However, on DAP/DNA we can only operate online profiler as branch instruction CPU interruption handler. As the future work Hardware assist mechanism to generate hardware branch history table is also considered to reduce the profiler's overhead.

Also the decision to select the hot path depends on the count number threshold. More study to choose right count number threshold is required.

About online synthesis, we are enabled to mapping the control instruction on DNA by token selection to DAP/DNA which use by data flow. Moreover, we could send the initial value of a register to DNA, and enabled to return the execution result in DNA to a register by preparing the stack area of register.

We are going to study the software/hardware technique for embodying a SysteMorph concept more effectively based on the result of this prototyping.

Acknowledgment

This research is based on MEXT (Ministry of Education, Culture, Sports, Science and Technology) intelligent cluster promotion project "Fukuoka intelligent cluster promotion project".

References

[1] Takanori Hayashida, Yusuke Kazu, Kazuaki Murakami, "SysteMorph : Functionality Morphing," The 4th Summer Workshop on Embedded System Technologies (SWEST4), 2002.

[2] IPflex, Inc. homepage, http://www.ipflex.com/

[3] Takanori Hayashida, Kazuaki Murakami, "Evaluating online Hot Instruction Stage Profilers for Dynamically Reconfigurable Functional Units," IEICE Trans. Information and Systems, Vol.E86-D, No.5, pp.901-909, May 2003

[4] Bala, E. Duesterwald, and S. Banerjia. Dynamo: A Transparent Dynamic Optimization System. ACM SIGPLAN Notices,35(5):1-2,2000

[5] T.Ball. Efficiently Counting Program Events with Support for On-line Queries. ACM Transactions on Programming Languages and Systems(TOPLAS),16(5):1399-1410, 1994