Evaluating Online Hot Instruction Sequence Profilers for DynamicallyReconfigurable Functional Units

Hayashida, Takanori
Graduate School of Kyushu University

Murakami, Kazuaki
Graduate School of Kyushu University

https://doi.org/10.15017/6046

出版情報：IEICE Transactions on Information and Systems. E86-D (5), pp.901-909, 2003-05. 電子情報通信学会
バージョン：
権利関係：

PAPER *Special Issue on Reconfigurable Computing*

Evaluating Online Hot Instruction Sequence Profilers for Dynamically Reconfigurable Functional Units

Takanori Hayashida[†], *Student Member and* **Kazuaki Murakami**[†], *Member*

SUMMARY Online profiling methodologies are studied for exploiting dynamic optimization. On a dynamic optimizable system with online profilers, it has to get accurate profile in early step of the program execution for effective execution. However, for getting more effective profile by online profiling, it has to satisfy "Rapidness" and "Accuracy". They are conflicted requirements. Therefore, it has to choose trade-off point at implementation. We focused into online Hot Instruction Sequence (HIS) profiler to exploit reconfigurable functional units. To circumstantiate the effectiveness of online HIS profiling, we build some evaluation models for experimental evaluation. Our profiler models are SC/DM, SC/FA and JC/DM. These models have different policy of event counting and table lookup. Our event counting policies are simple-counting or jumble-counting. On the other hand, table lookup policies are direct-map or full-associative. In our experimental evaluation, SC/FA and JC/DM models scored higher accuracy than SC/DM. The JC/DM model is able to implement by lower cost for table lookup, but it scored high accuracy comparable to SC/FA.

key words: online profiling, dynamically reconfigurable computing, hardware profiling

1. Introduction

Profiling is a very important technique for system optimization. The profile of a program tries to reflect or describe its behavior during execution.

Profiling technique can be classified into offline and online profiling. Offline profiling collects information of the whole program execution. After that, it analyzes all of the profile to extract the information for optimization. On the other hand, online profiling collects information about the execution of a program, analyses it, and extracts valuable information for optimization during program execution.

We are focusing into online profiling methodology for dynamic optimization through dynamically reconfigurable functional units (RFU). Online Hot Instruction Sequence (HIS) Profiler is our profiler that can exploit RFUs. HIS is the sequence of the most executed instructions and this online profiler tries to identify those sequences on-the-fly.

We chose a profiler model constructed by counters and tables among various profiler implementations. This model is very simple and it will be easy to realize by hardware. On this profiler model, we propose "Jumble Counting", it is new event-counting method for HIS profiling.

We give overview of online profiling in Sect.2. Next, we describe online hot instruction sequence (HIS) profiling in Sect.3. Section 4 evaluation models, methods for evaluation and details of experiment are presented. Section 5 gives results and discussion. We discuss some related works in Sect.6, and conclude in Sect.7.

Manuscript received September 18, 2002.
Manuscript revised November 18, 2002.
[†]Graduate School of Kyushu University

2. Online Profiling

Online profiling collects information about the execution of a program, analyses it, and extracts valuable information for optimization during program execution.

One of the advantages of online profiling is extraction of the dynamic behavior of the program. This dynamic behavior depends on input sequence. In many cases of offline profiling, some typical input sequences to analyze the program behavior is used. However, when the dynamic program behavior greatly differ according to the input sequence, it is difficult to analyze its behavior. In contrast, online profiling is able to analyze the dynamic behavior of the program due to the input sequence at that time.

Online profiling affects performance and energy consumption of the system because it takes the profile and analyzes it during program execution. In order to reduce the impact on performance it is convenient to build it on hardware.

In the case of online profiling by software, it will be effective dividing profiling and analysis to reduce the overhead on system performance. If both in parallel, it will cause a great performance reduction as the analysis process demands heavy calculation in many cases. Hence, it is effective that when the system is heavily loaded, the profiler does profiling only and analyzes it when the system less loaded.

In many cases, one cannot bear the increased energy consumption due to the addition of the profiler. However, it can decrease total energy consumption of the system if the system after reconfiguration/optimization obtained sufficient reduction of energy consumption, thus compensating the overhead of the profiler structure.

2.1 Requirement for Online Profiler

Since online profiler runs concurrently with the pro-

gram to be analyzed, it has to see about the following requirements[1].

- Low overhead

  In general, the profiler may not decrease system performance because purpose of the profiling is optimization of the system.

- Low cost

  If the profiler is embedded into the system, the final cost of the system is composed of the cost of the profiling mechanism plus the cost of the basic system. Here, cost includes both hardware cost and software cost. It's very important to reduce these costs as they influence performance, energy consumption and may increase design complexity of the system.

- Broad Applicability

  If a profiling method has broad applicability, it can extract the information for optimization from many kinds of profile information.

- Accurate Prediction

  As the analysis of profile information has to be done during program execution the profiler needs to predict the behavior of the program along execution. The effect of optimization depends on accuracy of this prediction.

Increasing accuracy normally conflicts with the final cost and overhead of the profiler as to be more accurate one needs to process and gather more information. Thus, there is a trade-off between the cost and accuracy of the online profiler, so one has to decide a point which satisfies both requirements.

2.2 Targets and Methodologies

There are following events are profiled by online profiling.

- Function Call
- Path Execution
- Branch Result
- Program Spot Execution
- Instruction Issue

Profiling is a methodology that extracts characteristics from a stream of these events. In many cases, online profiler profiles frequency of the events.

Function of an online profiler that profiles the frequency of event occurrence can be divided into the extractor and the notifier. The extractor extracts the frequency of the event from the given event stream by operating to the stream. On the other hands, the notifier notifies the extracted information by the extractor to an external optimizer.

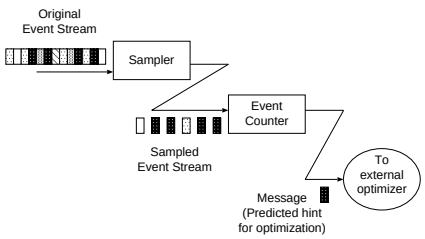

Fig. 1 Online profiler model.

The extractor's operations applied on the event stream are classified into sampling and event counting. Sampling reduces the search space of extraction by eliminating some occurred event from the original event stream. Characteristics extraction will be more easy than extraction from the original event stream. In contrast, information about eliminated event is lost: sampling sacrifices the accuracy of profile. Hence, sampling rate is a parameter that changes the accuracy of profile.

Event counting is an operation to extract occurrence frequency of each event. We assume an event counter contains a table in order to count the plural events independently. The ideal event counter structure have an infinite entry table and a counter for each entry. This counter is able to count all the occurred events independently without losing any information.

A real event counter cannot build such a table has, so it cannot count at the same time more events than entries it has.

When the number of events which may be occur is larger than number of entries of the table, there will be an entry replacement. The replacement policy is a parameter that affects the profile accuracy.

On the other hand, a fundamental method for realizing a notifier function is compare the event count and the notification threshold in every notification period. This method is applicable to both "hot (means occurs on high-frequency)" and "cold (means occurs on low-frequency)" events extraction. This notification threshold is a parameter that affects both entry replacement and prediction delay. Here, prediction delay means the time for notifying a message to the external optimizer.

Combining these operations, we can build a online profiler model shown in Fig.1.

This model includes two components having the following parameters:

- Sampler
  - Sampling rate (SR)
  - Sampling method
- Event Counter

– Number of entries (N)
– Notification threshold (TH)
– Replacement Policy

Using this model, we can define an arbitrary event counting based online profiler between the ideal "perfect event counting profiler" and the simplest sampler. The perfect event counting profiler extracts all events occurred more than threshold TH exactly. Denoting this profiler by our model, the parameters are $SR = 1, N = \text{inf}$. Entry replacement will never occur because the number of table entry is infinite. On the other hand, the simplest sampler on sampling rate SR is denoted by $TH = 0, N = 0$. There is two sampling methods periodic sampling and random sampling. Obviously, entry replacement will never occur on this profiler because there is no table in this profiler.

3. Online HIS Profiling for Dynamic RFU

3.1 Dynamically Reconfigurable Functional Unit

We assume a reconfigurable processor endowed with dynamically reconfigurable functional units(RFUs). RFU is a functional unit containing reconfigurable logic. Dynamically RFU is a RFU which can change its configuration on-the-fly.

We assume a dynamically RFU can perform any combination of several operations as a compound instruction by changing its configuration. The online profiling methodologies proposed in this paper exploit this DRFU by extracting hot instruction sequences during program execution.

3.2 Hot Instruction Sequence

Hot Instruction Sequence(HIS) is a sequence of instructions are the most executed in a program, i.e., sequence with the highest frequency of execution. To simplify the problem, in this paper we assume that an *instruction sequence* is a sequence in alignment with the time-axis of the opcode of a instruction word, and data dependency among operands of instructions in the sequence is not taken into consideration.

3.3 Methods applying Online HIS Profiling on dynamically RFU

Online HIS profiling methods can be classified whether restricting instructions are included in a sequence or not. There is a method to restrict the instructions by using the result of offline profiling, but we don't restrict them. We consider methods targeting the sequences include arbitrary instructions.

Furtuermore, as one of the other classifications, online HIS profiling methods are classified whether the support of variable length sequence is possible or not. Considering the hardware implementation of HIS profiler, variable length support will lead to increase in the complexity of implementation. In order to facilitate the hardware implementation only fixed length instruction sequence profilers are studied.

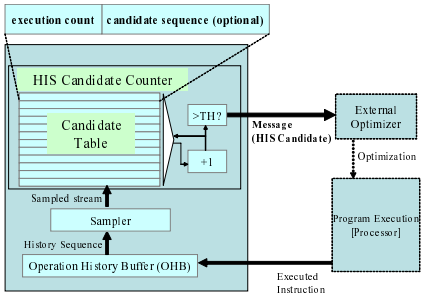

Fig. 2 Online HIS profiler model.

Exploiting the result of HIS profiling to optimization, we are assuming hardware offloading. Offloading the process of HIS to RFU described in Sect.3.1, will bring performance improvement. In this paper, we don't consider detail of optimization. We pay attention of only "accuracy" and "rapidness" of online HIS profiler.

4. Experiment and Result

4.1 Online HIS Profiler Models

In this paper, we use the online HIS profiler shown in Fig.2.

In this figure, Operation History Buffer (OHB) stores opcodes of several instructions executed just before. OHB is updated on every execution of an instruction.

Sampler sends a sequence of opcode stored in OHB to the HIS candidate counter differ to sampling rate. A sampler selects a sequence from OHB every constant number of instruction called *periodic sampler*. In contrast, a sampler works with a constant probability called *random sampler*.

HIS candidate counter is a table used for counting execution of instruction sequences. Generally, a tuple of the table will be ($executioncount$, $opcode\ sequence$). We assume the following counters:

- Simple Counter with a full-associative table
- Simple Counter with a direct-map table
- Jumble Counter with a direct-map table

Here, "Jumble Counter" is our counting method for online profiling. We also assume the models using "Simple Counter" is naive counting method for comparison. Details of these counters are below:

4.1.1 Simple Counter with a full-associative table (SC/FA)

When SC/FA counts execution of a sequence, it selects the entry by full-associative method. Each counter counts execution frequency of only one instruction sequence. An entry of the table is denoted by (*execution count*, *opcode sequence*).

When SC/FA received a sequence from the sampler, it lookups the table in order to find whether the sequence has already been counted or not. If the sequence was found in the table, it increments the counter of the entry. Contrary, when the sequence has not been found in the table, SC/FA decides an entry to replaced following LRU(least recently used) algorithm. After that, it replaces the entry by the current sequence and reset the counter.

When the counter value of an entry went over the notification threshold, SC/FA notifies the sequence to the external optimizer as a HIS candidate and clears the entry.

4.1.2 Simple Counter with a direct-map table (SC/DM)

When SC/DM counts execution of a sequence, it selects the entry by direct-map method. Each counter counts execution frequency of only one instruction sequence at a time. An entry of the table is denoted by (*execution count*, *opcode sequence*).

When SC/DM received a sequence from sampler, it calculates the hash value of the sequence to decide a table entry. After that, it compares current sequence with the sequence stored in the entry. If the sequences matched, it increments the counter of the entry. Contrary, when the sequences did not match, SC/DM replaces the entry by the current sequence and reset the counter.

When the counter value of an entry went over the notification threshold, SC/DM notifies the sequence to the external optimizer as a HIS candidate and clear the entry.

4.1.3 Jumble Counter with a direct-map table (JC/DM)

When JC/DM counts execution of a sequence, it selects the entry by direct-map method. Each counter counts the sum of execution frequency of multiple instruction sequences they having the same hash value. Consequently, this counter jumbles the execution count of different sequences. Since the execution count of sequences have the same hash value is jumbled, it is no need to memorize opcode sequence. So an entry of the table is denoted by (*execution count*).

When JC/DM received a sequence from the sampler, it calculates hash value of the sequence to decide a table entry. After that, it increments the counter of the entry. When the counter value of an entry went over the notification threshold, JC/DM notifies the current sequence to the external optimizer as a HIS candidate and reset the counter.

Table 1 Evaluation Models

Evaluation Model Name	Model	Sampling Rate	Threshold
SC/DM_1	SC/DM	1	512
SC/FA_1	SC/FA	1	512
JC/DM_1	JC/DM	1	512
SC/DM_{64}	SC/DM	1/64	8
SC/FA_{64}	SC/FA	1/64	8
JC/DM_{64}	JC/DM	1/64	8
SC/DM_{128}	SC/DM	1/128	4
SC/FA_{128}	SC/FA	1/128	4
JC/DM_{128}	JC/DM	1/128	4
SC/DM_{256}	SC/DM	1/256	2
SC/FA_{256}	SC/FA	1/256	2
JC/DM_{256}	JC/DM	1/256	2
SC/DM_{512}	SC/DM	1/512	2
SC/FA_{512}	SC/FA	1/512	2
JC/DM_{512}	JC/DM	1/512	2

4.2 Evaluation

4.2.1 Evaluation Models

In order to evaluate the accuracy of the online HIS profiler models described in Sect.4.1, we assign the parameters shown in Table1 for building evaluation models. Furthermore, as common parameters, sequence length is fixed to 4, number of entries of HIS candidate counter fixed to 1024 in this evaluation.

Moreover, notification threshold of each evaluation models are given by $max(2, TH_{base} \times SR)$. The reason for having set the minimum value of the notification threshold to 2 is to eliminate the infrequently executed sequences sampled by accident. When the threshold set to 2, it can prevent misprediction by such accident. In this evaluation, we fixed TH_{base} to 512.

Using these evaluation models, we can evaluate the effects to accuracy of profile by changing the sampling rate or/and the counting policy.

4.2.2 Evaluation Measure

In order to evaluate these profiler models, we use $Hitrate$ and $Noiserate$ based on [2].

In below definitions, s is an arbitrary instruction sequence, and S is an arbitrary set of instruction sequence. T_{fin} shows finish time of program execution.

$$freq(s,t) = \{execution\ count\ of\ s\ until\ t\} \quad (1)$$

To simplify notations, total execution count of s is noted $freq(s)$. Obviously, $freq(s)$ equals to $freq(s, T_{fin})$.

The sum of execution count of instruction sequences in a set S is noted

$$freq(S,t) = \sum_{s \in S} freq(s,t) \tag{2}$$

In the same way as $freq(s)$, $freq(S)$ equals to $freq(S, T_{fin})$.

HIS_{th} denotes a sequence belongs to the "True" HIS set.

$$HIS_{th} = \{s : freq(s) > th\} \tag{3}$$

$D(t)$ denotes an instruction sequence set constructed with the detected instruction sequence. Using these notations, $Hit(t)$, $Noise(t)$, $Hitrate(t)$ and $Noiserate(t)$ are defined by the following equations.

$$Hit(t) = D(t) \cap HIS_{th} \tag{4}$$

$Hitrate(t)$

$$= \frac{\sum_{s_i \in HIT(t)} \{freq(s_i) - freq(s_i, dd_i)\}}{freq(HIS_{th})} \tag{5}$$

Here, dd_i is the *detection delay* of s_i. dd_i means detection time of s_i. Accordingly, $freq(s_i, dd_i)$ means execution count of s_i before s_i is detected by the profiler.

As shown the equation, $Hitrate$ will be higher if the detected sequences are "right" (means that they are included in "True HIS"). Moreover, if the right detection occurs in early step of program execution, $Hitrate$ will be more higher .

Like $Hitrate$, $Noiserate$ is denoted by the following equations.

$$Noise(t) = D(t) - HIS_{th} \tag{6}$$

$Noiserate(t)$

$$= \frac{\sum_{s_i \in Noise(t)} \{freq(s_i) - freq(s_i, dd_i)\}}{freq(HIS_{th})} \tag{7}$$

$Noiserate$ denotes the rate of mispredicted sequences included in the message. Reducing $Noiserate$, reduces the wrong sequences included in the message, or enlarges the prediction delay of the wrong sequences.

4.3 Experiment

We experimented to evaluate the accuracy of each online HIS profiler models. Environment of evaluation experiment is shown in Table2.

In this evaluation experiment, we implemented the online HIS simulator in C code. The input of the simulator is the instruction trace generated by SimpleScalar simulator. We calculated $Hitrate$ and $Noiserate$ from the message sequence output from our simulator. We use the 300M instruction trace on top of each program. Furthermore, we defined the "True" HIS is the set of sequence executed more than 0.1% of total executed instructions (=300M in this experiment). This threshold was decided by preparatory experiment, the threshold gives the "True" HIS set includes the instruction sequences frequently executed. Also, its number of instances are enough smaller than others. Moreover, we use the hash function by XOR and SHIFT operation to converted value from the instruction sequence, it makes flat for calculated hash value with considering the frequency of them.

Table 2 Environment of experiment

Instruction set	SimpleScalar 2.0[9]
Compiler	gcc -O2
Target Program	SPECCPU2000 Integer Benchmark Program set[10] 164.gzip, 192.parser, 256.bzip2
Input Sequence	train (from default SPECCPU2000 set)
Execution Area	300M instructions from top.

4.4 Result and Discussion

The results of the experiment are shown in Fig.3–12. Figure 3–8 show differences of $Hitrate$ and $Noiserate$ by changing counter policy. The sampling rate is fixed to 1 in these figures in order to discuss the upper bound of $Hitrate$ and $Noiserate$. First, we observe the effect of replacement policy. From Fig.3,5 and 7 in respect of only $Hitrate$ SC/FA_1 has the best accuracy among these profilers. SC/DM_1 scored the worst $Hitrate$ among them. However, if we focus on the $Noiserate$, Fig.4,6 and 8 shows the SC/FA_1, scoring the best $Hitrate$, scores largest $Noiserate$ for all experimented benchmark programs.

Both $Hitrate$ and $Noiserate$, JC/DM_1 gave very interesting result. In all benchmark programs, JC/DM_1 scored a good $Hitrate$, comparable to SC/FA_1. In contrast, JC/DM_1 scored a lower $Noiserate$ than SC/FA_1 at 164.gzip (Fig.4) and 197.parser (Fig.6). At 256.bzip2, all models scored similar $Noiserate$ (Fig.8).

Comparing the lookup cost of the table in the profiler, full associative table implementation has high cost either in hardware or in software. When the number of entries of the table is greater, the complexity of the full associative table will greatly increase. Compared to the lookup cost, direct mapped table is easy to implement. But the result of SC/DM_1 is not good on $Hitrate$.

In these result, JC/DM is a very good model of online HIS profiler because its score is comparable to SC/FA, and the cost for table lookup is comparable to SC/DM's one.

Since JC/DM_1 scored good $Hitrate$ and $Noiserate$, we then focus on changing the sampling rate for JC/DM

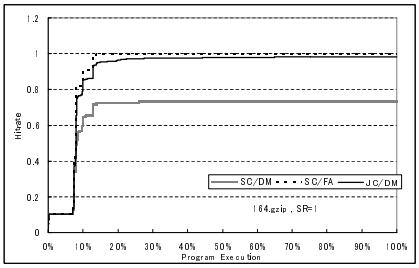

Fig. 3 Hitrate by counter policy (164.gzip)

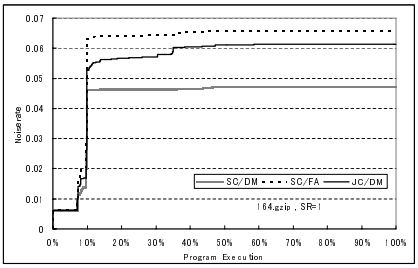

Fig. 4 Noiserate by counter policy (164.gzip)

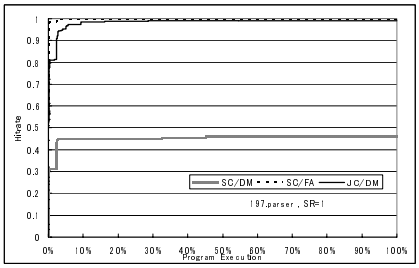

Fig. 5 Hitrate by counter policy (197.parser)

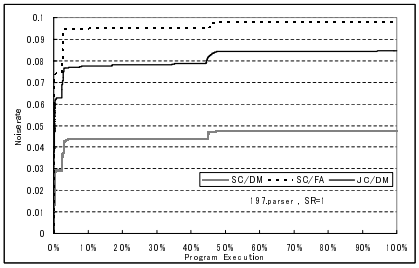

Fig. 6 Noiserate by counter policy (197.parser)

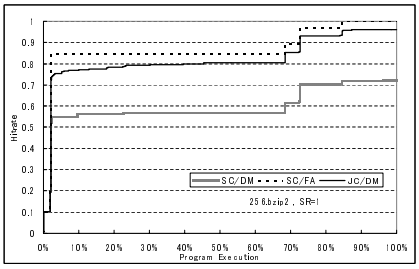

Fig. 7 Hitrate by counter policy (256.bzip2)

model.

Figure 9–12 shows the *Hitrate* and *Noiserate* of the JC/DM model at 164.gzip and 197.parser parameterized by its sampling rate.

For these programs, the sampling rate does not cause large difference to *Hitrate* and *Noiserate*. In Fig.11, JC/DM_{512} scores a low *Hitrate* at the exceedingly early time of program execution. However, after 3% of the program executed, its *Hitrate* is the same as others. Consequently, JC/DM_{512} is not worse than others, because it has enough rapidness.

In Fig.10 and 12, JC/DM_{512} scores lower *Noiserate* than others including JC/DM_1. Probably, this is caused by the difference of $(TH \times SR)$. JC/DM_{512} is equivalent to the JC/DM model with these parameters: $TH_{base} = 1024, SR = 512$. It is easy to predict that the higher TH_{base} model will score lower *Noiserate* to similar HIS sets. We need to change the parameter TH_{base} to clear this problem; this is a future work.

Regarding JC/DM_{512}, all the results of Fig.9–12 are similar to each other. This shows that the JC/DM profiler can keep its accuracy even if the sampler has lost some information. Also, it brings an easy implementation of the hardware HIS profiler because it is no need to the table replacement and the counter increment for every instruction execution. A JC/DM based profiler has to only calculate the hash values and counter increment. The profiler has to finish the operation for an instruction sequence before arrival of the next sequence. It relaxes the time constraint to the profiler implementation.

In this experiment, we considered about *Hitrate* and *Noiserate* to evaluate accuracy and rapidness. These results show that our online HIS profilers can generate enough accurate HIS candidate set rapidly, and probably they have time margin for the table accessing. Evaluation about impact to the system performance and the energy consumption is our future work.

5. Related Work

Various methodologies of online profiling and dynamic optimization are investigated in recent years. However, the events of profiling target of the most of them are execution of particular instructions (i.e. branch instruction),[3],[6], [8], program execution paths [2],[7], or threads [5], load values [1] and so on. These profile is dependent on the location in the target program. They are different from our work, our profile information is independent the location (or address).

Sastry *et al.* proposed a profiling methodology by hardware called *Stratified Sampling* in [1], and evaluated their method on *Load Value Profiler*. Our method is similar to their method includes sampler, counter and hash table. They proposed a table replacement algorithm using a finite state machine. Our replacement algorithm is more simple, but we can get high accuracy on online HIS profiling. We have to investigate whether

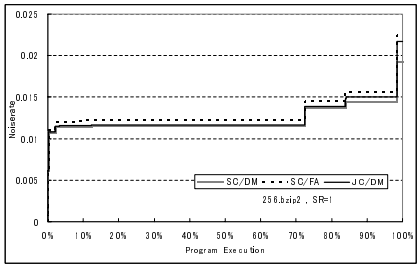

Fig. 8 Noiserate by counter policy (256.bzip2)

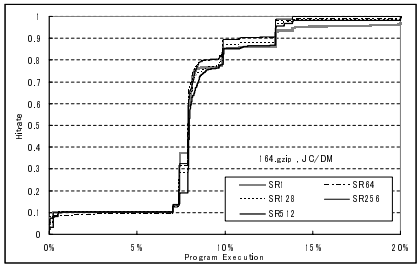

Fig. 9 Hitrate by sampling rate on JC/DM (164.gzip)

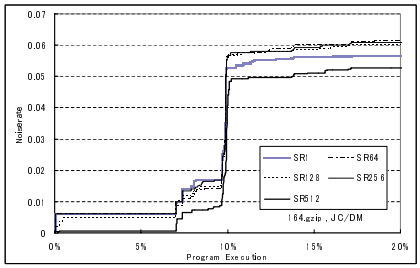

Fig. 10 Noiserate by sampling rate on JC/DM (164.gzip)

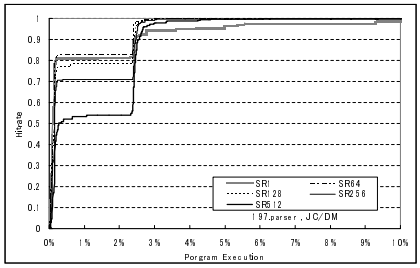

Fig. 11 Hitrate by sampling rate on JC/DM (197.parser)

the more complex algorithm like [1] will impact to accuracy and rapidness of HIS profiling.

In [2], Duestenwald *et al.* proposed the Next Execution Tail(NET) prediction to predict hot (means frequently executed) path rapidly. They showed that the NET Prediction performs better than a naive path profiling prediction, giving great speedup for prediction and keeping its accuracy. Their path counting method counts the frequency of only the head of a path. Then it predicts the hot path candidate by first executed path after over the threshold. It is similar to our Jumble Counting, but our method has more flexibility by changing the hash function.

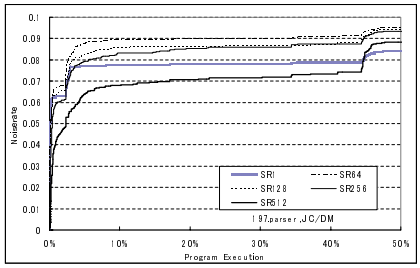

Fig. 12 Noiserate by sampling rate on JC/DM (197.parser)

Online profiling and dynamic optimization methodologies for Java byte-code are studied in [5],[6]. Arnold *et al.*[5] proposed the profiling methodology to find the threads are frequently executed. Their method needs support of Java Virtual Machine (JVM) to find the"Hot" threads, it is similar to ours. Our profiler assumes hardware implementation. However, granularity of profile [5] is grater than ours, because their method is triggered by changing thread, but our trigger is an instruction execution at minimum. Software profiling method for dynamic inlining of Java byte-code is proposed by Suganuma *et al.* [6]. Their methodology collect the profile about "Hot" function call. Realizing the profiling in software, their method uses *dynamic instrumentation*. Our method assumes simultaneously profiling with program execution.

su Moreover, the Code Morphing System (CMS) for Crusoe^{TM}processor by Transmeta Corporation [7] is one of the famous dynamic software optimization systems. CMS profiles the execution frequency of program blocks by instrumentation. Hence, the collected profiles dependent with location in the target program. It is different from our methodology. But the CMS does not use precedently profiled information for profiling and optimization. It is similar to our methodology. Also, the CMS profiles the execution path include some branches simultaneously. Both information about block execution and path execution are used at optimization phase. Our profiling method focuses only one event stream, it is different from ours.

[4] is not only frequently event profiling, but also profiles the change of program behavior dynamically. Dhodapkar *et al.* proposed the methodology regarding that the set of accessed address by program will be change the fraction of program. Because changing of the set means changing of the program behavior. Our methodology profiles the sequence of instructions, HIS will be changed by program behavior, so their method can apply for improve our method.

6. Conclusion

In this paper, we evaluated online HIS profilers constructed by samplers and counters. Our result showed they have enough accuracy for detecting HIS. Par-

ticularly, Jumble Counting with Direct Map table model(JC/DM) that needs lower lookup cost, scored high $Hitrate$ comparable to profiler using full associative table.

These results are promising, motivation the use of this profiler in the development of on our dynamically reconfigurable system. In this paper we considered only the accuracy of the profiler. Analyzing the performance of the profiler for different compositions of profiled sequences and dependencies if scheduled as future work. Also we need to evaluate the overhead of this profiling mechanism in real dynamically reconfigurable units. It brings the evaluation of the performance improvement and impact to energy consumption.

Acknowledgement

We would like to thank the members of Yasuura-Murakami-Matsunaga Lab. at Kyushu University for their discussion and comments. We also would like to thank the anonymous reviewers for their helpful comments. This work has been supported by the Grant-in-Aid for Scientific Research (KAKENHI) of the Japan Society for the Promotion of Science (JSPS), under Grant (A)(2) 12358002 and (A)(2)13308015. We are grateful for their support.

References

[1] S.S.Sastry, R.Bodik and J.E.Smith, "Rapid Profiling via Stratified Sampling," Proc. of IEEE the 28th Annual International Symposium on Computer Architecture(ISCA 2001), pp.278–289, Jul,2001.

[2] E.Duestenwald and V.Bala,"Software Profiling for Hot Path Prediction: Less is More," Proc. of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems(ASPLOS 2000), pp.202–211, Nov.,2000.

[3] C.B.Zilles and G.S.Sohi, "A Programmable Co-Processor for Profiling," Proc. of 7th International Symposium on High Performance Computer Architecture(HPCA 2001), pp.241–252, Jan.,2001.

[4] A. S. Dhodapkar and J. E. Smith, "Managing Multi-Configurable Hardware via Dynamic Working Set Analysis," The 29th Annual International Symposium on Computer Architecture(ISCA 2002), May,2002.

[5] M. Arnold, S. Flink, D. Grove, M. Hind and P. F. Sweeney, "Adaptive Optimization in the Jalapeño JVM," Proc. of Object–Oriented Programming Systems, Languages and Applications (OOPSLA2000), pp.47–65, Nov.,2000.

[6] T. Suganuma, T. Yasue and T. Nakatani, "An Empritical Study of Method In-lining for a Java Just-In-Time Compiler," 2nd Java Virtual Machine Research and Technology Symposium(JVM'02), Aug.,2002.

[7] A. Kaliber, "The Technology Behind Crusoe^{TM} Processors," Transmeta Corporation WhitePaper, Jan.,2000.

[8] M. Merten, A. R. Trick, E. M. Nystorm, R. D. Barnes and W. W. Hwu, "A Hardware Mechanism for Dynamic Extraction and Relayout of Program Hot Spots," Proc. of The 27th Annual International Symposium on Computer Architecture(ISCA2000), pp.59–70, Jun.,2000.

[9] SimpleScalar Toolset Homepage, *http://www.simplescalar.com/*

[10] Standard Performance Evaluation Corporation (SPEC) Homepage, *http://www.spec.org/*