Performance Evaluation of a Reconfigurable Instruction Set Processor

Mehdipour, Farhad
Research Institute for Information Technology, Kyushu University

Noori, Hamid
Institute of Systems, Information Technologies and Nanotechnologies

Honda, Hiroaki
Institute of Systems, Information Technologies and Nanotechnologies

Inoue, Koji
Department of Informatics, Kyushu University

他

出版情報：Proceedings of International SoC Design Conference. 2008, pp.184-187, 2008-11-25
バージョン：
権利関係：

Performance Evaluation of a Reconfigurable Instruction Set Processor

Farhad Mehdipour
Research Institute for Information Technology, Kyushu University, Japan
farhad@c.csce.kyushu-u.ac.jp

Hamid Noori, Hiroaki Honda
Institute of Systems, Information Technologies and Nanotechnologies, Japan
{noori, dahon}@c.csce.kyushu-u.ac.jp

Koji Inoue, Kazuaki Murakami
Department of Informatics, Kyushu University, Japan
{inoue, murakami}@i.kyushu-u.ac.jp

***Abstract*—Performance evaluation is a serious challenge in designing or optimizing reconfigurable instruction set processors. A combined analytical and simulation-based model (CAnSO∗) is proposed and validated for performance evaluation of a typical reconfigurable instruction set processor. The proposed model consists of an analytical core that incorporates statistics gathered from cycle-accurate simulation to make a reasonable evaluation. CAnSO has clear speed advantages and compared to cycle-accurate simulation, it proves almost 2% variation in the speedup measurement.**

Keywords- Reconfigurable instruction set processors; analytical modeling; performance evaluation

I. INTRODUCTION

Reconfigurable instruction set processors (RISPs) introduce an effective approach for implementing embedded systems similar to application-specific instruction set processors (ASIPs) and extensible processors. A RISP mainly consists of a microprocessor core that is extended with a reconfigurable accelerator (RAC) [1] The base processor (BP) implements non-critical parts of the applications and reconfigurable accelerator is responsible for executing critical parts. Fig. 1 shows a general template of a RISP including a baseline processor (BP), a reconfigurable accelerator (RAC), which is based on a matrix of functional unit (FUs), and a configuration memory. Hot portions of applications are identified and used for generating CIs. CIs are mapped onto the RAC and then configuration's bit-stream is generated for each CI and stored in the configuration memory prior to application execution. The associated bit-stream is loaded on the RAC while encountering a CI and then the CI is executed on the RAC at runtime [4][5].

Performance evaluation of a RISP challenges both the designing of such system and optimizing an existing one for an objective function. In either event, a designer is interested in obtaining optimum system configuration and therefore needs to perform a performance analysis in terms of the performance metrics e.g. speedup, area, energy consumption and etc.

The main contribution of this paper is to introducing and validating an analytical model for performance evaluation of a reconfigurable instruction set processor. Even though the core is an analytical model, it utilizes trace-driven information e.g. miss-events' rates and the latency of executing CIs on the base processor to emphasize on the application aspect and provide a reasonably accurate evaluation.

II. A COMBINED ANALYTICAL AND SIMULATION-BASED MODEL (CANSO)

The design or optimization of a reconfigurable accelerator requires a considerable time in exploring the design space and finding a proper architecture meeting the design constraints and gaining a desirable performance. Although high performance computers can be used for cycle-accurate simulation of the workloads in a shorter time at these days, exploring a large design space is still both human and computationally intensive [3]. This motivates a performance evaluation approach based on computationally simple analytical model. Relying only on the analytical model may not be able to precisely take the effect of realistic factors of application behavior including miss-events (e.g. data and instruction cache misses or branch miss-predictions) into account. Therefore, the proposed model (CAnSO) also has a qualitative insight in providing trace-driven information from the application by means of cycle-accurate simulation of applications.

Fig. 2 depicts how a combined model is generated and then utilized for performance evaluation. In the first step, all the demanded applications are simulated using a cycle-accurate simulator and required information (e.g. the number of CIs, execution frequency of CIs and so on) are collected. Afterward, the information is used for simplifying and calibrating the proposed pure analytical model to make it more precise and empirical. After establishing the model, CAnSO can be used for performance evaluation of each application. Again the statistics gathered from the corresponding application can be utilized for more accurate estimation of the performance.

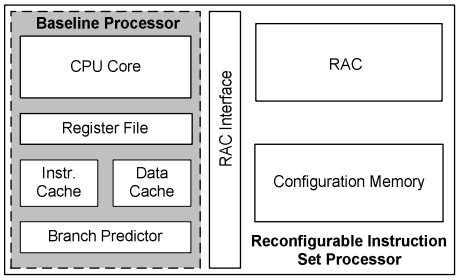

Fig. 1. A reconfigurable instruction set processor's general template

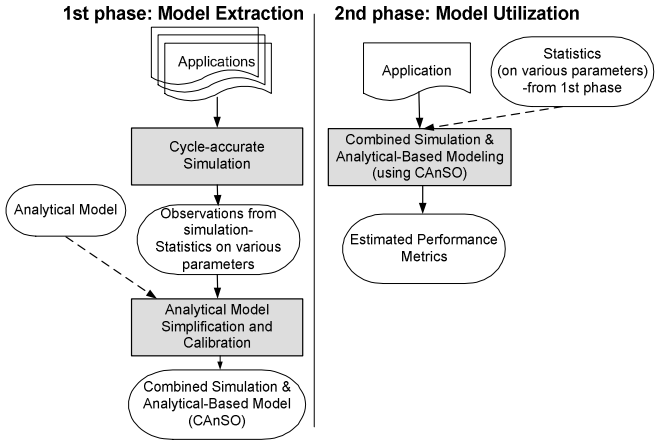

Fig. 2. Model extraction and utilization phases

∗ CAnSO in Japanese means simple.

One usage of such model is in design space exploration of the accelerator where designer intends to study the effect of changes in the architectural specifications (e.g. size, dimensions and etc.) of the accelerator. It substantially shortens the exploration time with a reasonable accuracy.

III. THE ANALYTICAL MODEL

A. Basic Model Definitions

In our template architecture (Fig. 1), the BP is an in-order general five-stage RISC processor and RAC is a coarse-grained tightly-coupled reconfigurable hardware which implements custom instructions. To access required operands in the RAC, the content of all registers are shared between RAC and conventional FUs [1]. Controlling configurations i.e. the task of loading and initiating the RAC which can be hardware or software-based. In a software-based mechanism starting address of a CI is replaced with a special instruction [4]. To implement a hardware-based method, starting address of each CI and index to the configuration memory is stored in a content addressable memory (CAM) for fast retrieval of the corresponding bit-stream from the configuration memory. Below our terminology and some definitions are presented:

n_{tcc}: Total number of clock cycles spent for executing an application on the base processor including entire miss-events.

n_{CI} : Total number of CIs

l_i : The number of primitive instructions in CI_i. Instructions belong to the BP's instruction set architecture (ISA).

τ_{BP}^{i} : The time required for execution of equivalent sequence of instructions corresponding to CI_i on the BP in term of the number of clock cycles.

τ_{RAC} : A fixed-delay for executing a CI on the RAC.

τ_{OVH} : Overhead time including RAC reconfiguration time and communicating between RAC and BP.

$\theta_i = (\theta_{i1}, \theta_{i2}, \theta_{i3}, ..., \theta_{i,m_i})$, where θ_{ij} is the frequency of j^{th} occurrence of CI_i and m_i is the number of times that CI_i is executed during the application run-time. Correspondingly, $O_i = \sum_{j=1}^{m_i} \theta_{ij}$ is the total number of executions of CI_i.

Two type of execution are supposed for CI_i:

Single execution: in the j^{th} occurrence of CI_i, it is executed once ($\theta_{ij} = 1$). In Fig. 3, second occurrence of CI_1 and first occurrence of CI_2 are the single executions ($\theta_{12} = 1$, $\theta_{21} = 1$).

Continuous execution: in a continuous execution, CI is repeatedly executed ($\theta_{ij} > 1$), and the RAC is configured at the first execution and operates without need to reconfiguration for the next executions of the same occurrence. Fig. 3 shows that CI_1 and CI_2 in their first and second occurrences are executed continuously ($\theta_{11} = 2$, $\theta_{22} = 3$). According to two above definitions, S_i and C_i are introduced which contain the index of intervals of single and continuous executions of CI_i: $S_i = \{j \mid j \in \{1,...m_i\}, \theta_{ij} = 1\}$ and $C_i = \{j \mid j \in \{1,...m_i\}, \theta_{ij} > 1\}$.

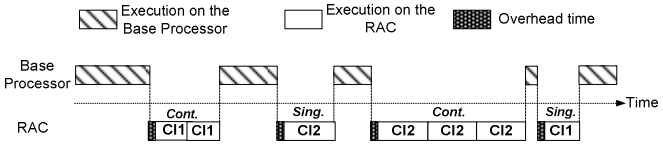

Fig. 3. Various types of CIs execution on the reconfigurable accelerator

s_i : Denotes partial speedup for CI_i and s_{ij} indicates the partial speedup achievable via j^{th} execution of CI_i.

$$s_{ij} = \frac{\tau_{BP}^{i}}{\tau_{RAC} + \tau_{OVH}} + \left(\frac{\tau_{BP}^{i}}{\tau_{RAC}} \times (\theta_{ij} - 1) \right) \quad (1)$$

The speedup for CI_i can be calculated as:

$$s_i = \sum_{j=1}^{m_i} s_{ij} = m_i \times \left(\frac{\tau_{BP}^{i}}{\tau_{RAC} + \tau_{OVH}} \right) + \sum_{j=1}^{m_i} \left(\frac{\tau_{BP}^{i}}{\tau_{RAC}} \times (\theta_{ij} - 1) \right) \quad (2)$$

B. Speedup Formulation

The fraction of application execution time which is transferred to the RAC and the fraction of application running on the BP are as Eq. 3 and 4, respectively. According to the Amdahl's law, the overall speedup (s_o) can be calculated as Eq. 5:

$$f_{RAC} = \frac{\sum_{i=1}^{n_{CI}} \left(\tau_{BP}^{i} \times O_i \right)}{n_{tcc}} \quad (3) \qquad f_{BP} = \frac{\left(n_{cc} - \sum_{i=1}^{n_{CI}} \left(\tau_{BP}^{i} \times O_i \right) \right)}{n_{tcc}} = 1 - f_{RAC} \quad (4)$$

$$s_o = \frac{n_{tcc}}{\left(n_{tcc} - \sum_{i=1}^{n_{CI}} \left(\tau_{BP}^{i} \times O_i \right) \right) + \psi(\theta, \tau)} \quad (5)$$

$$\psi(\theta, \tau) = \sum_{i=1}^{n_{CI}} \left(\sum_{j \in S_i} \left(\theta_{ij} \times (\tau_{RAC} + \tau_{OVH}) \right) + \sum_{j \in C_i} \left((\tau_{RAC} + \tau_{OVH}) + \left((\theta_{ij} - 1) \times \tau_{RAC} \right) \right) \right) \quad 6)$$

C. The Effect of CI Length

In the CI generation phase, some CIs might be generated which contain more instructions than the number of resources available in the RAC. Dividing larger CIs to a number of smaller CIs through a temporal partitioning algorithm is one solution. When $l_i > n_{FU}$, supposing $L = \{k \mid k \in \{1,...,n_{CI}\}, l_k > n_{FU}\}$ and $P = \{p_k \mid k \in L, p_k = \left\lceil \frac{l_k}{n_{FU}} \right\rceil\}$ each $CI_k (k \in L)$ is divided to p_k smaller CIs. A large CI's execution irrespective of whether it is single or continuous is divided to a number of single executions; therefore execution of each partition of the CI necessitates a reconfiguration. It means for CI_k :

$$m'_{k \in L} = O_i \times p_k, m'_{k \notin L} = m_i$$
$$\theta'_{k \in L} = ((1,...,1),(1,...,1),...,(1,...,1)), |\theta'_{k \in L}| = m'_{k \in L}, \theta'_{k \notin L} = \theta_{k \notin L}$$
$$S'_{i \in L} = \{1,...m'_i\}, S'_{i \notin L} = S_i \text{ and } C'_{i \in L} = \varnothing, C'_{i \notin L} = C_i \quad (7)$$

D. Side-Effects

Control Instructions: In case of inclusion of control instructions in the CIs, the rate of miss-predicted branches might be reduced

which results in increase in the speedup [4]. δ_{bm} is defined as variation in branch miss-predictions and p_{bm} as a number of penalty cycles imposed by a branch miss-prediction (Eq. 8). δ_{bm} is negative when the number of miss-predictions reduces, otherwise it is positive. As aforementioned, n_{tcc} encompasses all miss-events as well as branch miss-predictions.

Instruction Cache Misses: The similar influence can be seen when the impact of instruction cache misses are taken into account. Since, the execution of CIs is performed on the RAC without need for fetching instructions belonging to the CIs, the rate of accesses to instruction cache and therefore instruction caches misses might be reduced [4]. It is assumed that the instruction cache misses reduction is δ_{im} . This implies that the fraction of time concerning to BP reduces and speedup rises not only because of execution of CIs on the RAC, but also due to reduction in instruction cache's miss rate. Eq. 8 is introduced for overall speedup calculation and includes the side-effects as well:

$$s_O = \frac{n_{tcc}}{\left(n_{tcc} - \sum_{x=\{b,i\}} \delta_{xm} \times p_{xm} + \sum_{i=1}^{n_{DI}} \left(\tau_{BP}^{i} \times \mathrm{O}_i \right) \right) + \psi'(\theta', \tau)} \quad (8)$$

E. RF's Input/Output Ports

Register file is shared between BP and RAC in the template architecture. It is assumed that Δ_{reg} / ∇_{reg} are the number of read/write ports of the register file, and $\Delta_{CI}^{i} / \nabla_{CI}^{i}$ are the number of inputs and outputs of the CI_i. Additional clock cycles for reading/writing from/to the register file should be taken into account when the number of inputs/outputs of the CI is greater than the available number of register file's read/write ports. The overhead time increases as Eq. 9.

$$\tau'_{OVH} = \tau_{OVH} + \max(0, \left\lfloor \frac{\Delta_{CI}^{i} - \Delta_{reg}}{\Delta_{reg}} \right\rfloor) + \max(0, \left\lfloor \frac{\nabla_{CI}^{i} - \nabla_{reg}}{\nabla_{reg}} \right\rfloor) \quad (9)$$

F. RAC's Delay

The RAC's delay is an essential element regardless of whether the analytical or simulation approaches are used for performance evaluation. A simple approach is introduced that estimates the latency of RAC's critical path by means of analyzing the RAC's structure and using the delays of RAC's basic components (obtained through their synthesis) comprising functional and routing resources.

In our template architecture, RAC_h^w includes a matrix of FUs with width equal to w and height equal to h and basically has a combinational logic (Fig. 4). Each FU implements an instruction level operation. Routing resources are available from each FU in a row to FUs in consecutive row and also to adjacent FUs at the same row (dashed upward line) (Fig. 4).

It is assumed that all FUs in the RAC's architecture implement similar operations and have the same functionality and latency (i.e. $\forall i, j \quad \tau_{FU}{}_{j}^{i} = \tau_{FU}$). Each mux in row i receives all outputs of the FUs in upper rows and also from its adjacent FUs at the same row. Furthermore, the total number of muxes in row i is equal to the number of FUs in $(i+1)$th row (which is n_{i+1}) multiplied by two due to existing two input sources for each FU, however, we use the same indices for two muxes of a FU. Consequently, critical path delay of RAC_h^w can be calculated as:

$$\tau_{RAC_h^w} = \sum_{i=1}^{h} \tau_{FU} + \sum_{i=1}^{h-1} \tau_{MUX_i^k}, \quad k \in \{0,1,\ldots,w\} \quad (10)$$

where, $\tau_{mux}{}_{j}^{i}$ is jth mux between rows i and $i+1$. In Eq. 8, $\psi'(\theta', \tau)$ is replaced with $\psi'\left(\theta', \tau_{RAC_h^w} + \tau_{OVH}\right)$.

IV. Simplification and Calibration

The proposed analytical model can be simplified and calibrated according to the following observations:

1. In our template architecture, control instructions are not supported and are excluded from CIs, therefore, there is no reduction in branch miss-prediction.
2. Loading configurations from the configuration memory without need for fetching instructions from instruction cache results in reduction in instruction cache accesses as well as cache misses. Our experiments depict that the average reduction in access to instruction cache is almost 17% and in cache misses is almost 3%.
3. It is assumed that the ratio of single to continuous execution for each application is α . The average value for α is almost 43%. Putting altogether, following equations would be obtained in which, all variables marked with the * are obtained through simulation in the model extraction phase (Fig. 2).

$$s_O = \frac{n_{tcc}^{*}}{\left(n_{tcc}^{*} - \delta_{im}^{*} \times p_{im}^{*} - \sum_{i=1}^{n_{DI}^{*}} \left(\tau_{BP}^{i} \times \mathrm{O}_i^{*} \right) \right) + \psi'\left(\theta', \tau_{RAC_h^w} + \tau_{OVH}\right)^{*}} \quad (11)$$

$$\psi'\left(\theta', \tau_{RAC_h^w} + \tau_{OVH}\right) = \sum_{i=1}^{n_{CI}^{*}} \mathrm{O}_i^{*} \times \left(\alpha \times \tau'_{OVH} + \tau_{h\ RAC}^{w} \right)$$

V. Experiments

A. Experimental Setup

A reconfigurable instruction set processor matching to the general template in Fig. 1 is assumed. Fourteen applications of Mibench [2] from various domains (e.g. automotive, security, consumer, network, telecommunication) are used for generating CIs (DFGs) similar to the approach in [4]. Simplescalar's cycle accurate simulator [4] has been extended to simulate a reconfigurable instruction set processor. Firstly, our analytical model is established according to the first phase in Fig. 2. All attempted applications are simulated and required information (variables superscripted with the * in Eq. 11) are collected. It takes almost four hours to completion (on a PC Dual Core, Intel 6600@2400Mhz, 2GB RAM). Next, the model simplification and calibration is accomplished. The RAC structure comprises sixteen FUs locating in five rows including six, four, three, two and one, respectively (similar to a RAC in [4]). Then, the delay of RAC ($\tau_{RAC}{}_{h=5}^{w=6}$) is calculated using the approach in Section III.F.

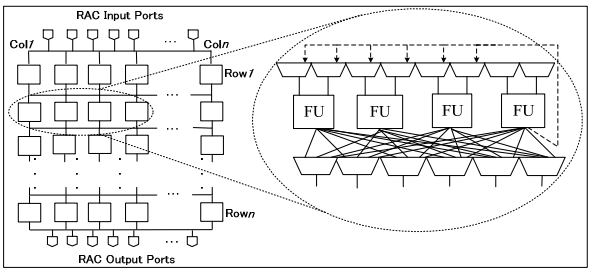

Fig. 4. Assumed RAC architecture

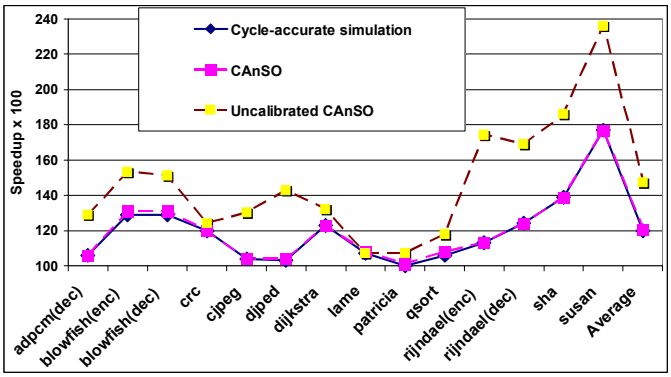

Fig. 5. CAnSO tracks the results of cycle-accurate simulation while uncalibrated CAnSO results in 22% difference in average

B. *Model Validation*

Fig. 5 demonstrates CAnSO successfully tracks cycle-accurate simulation. This accuracy is gained because of incorporating the information of a trace-driven simulation in the model extraction phase. We studied the effect of constructing analytical model based on the information collected from cycle-accurate simulation on the accuracy of the results. Ignoring some realistic information from cycle-accurate simulation e.g. single or continuous executions frequencies and statistics on miss-events as well substantiates the efficacy of the CAnSO. According to Fig. 5, the achieved speedup using uncalibrated CAnSO (in which some simulation information are ignored) differs 22% in average with cycle-accurate simulation, while the average variation of CAnSO and cycle-accurate simulation is less than 2%. Moreover, uncalibrated CAnSO does not successfully track the simulation approach for some applications (e.g. *djpeg* and *rijndael(enc)*) due to essential impact of the ignored parameters.

C. *Design Space Exploration Using CAnSO*

Designing an appropriate RAC for a reconfigurable instruction set processor is a challenging issue. The CAnSO is suitable when designer intends to explore a large design space. For instance, the design of a RAC including different components entails a multitude of design parameters [3]. DSE could be very time consuming even in case of simulation. For instance, examining 100 design points and fourteen input applications by means of simulation takes almost seventeen days while using CAnSO, it reduces to almost four hours. Consequently, re-simulating applications is not needed thus required time and efforts for extracting model are alleviated. Fig. 6 shows the speedup variation with respect to different RAC dimensions for three randomly selected applications of Mibench [2].

VI. CONCLUSION

An analytical model for speedup evaluation of a reconfigurable instruction set processor was proposed. To become more accurate and realistic, CAnSO is established and calibrated based on statistics gathered from cycle accurate simulations of attempted applications. This model provides sufficient flexibility in a fast evaluation of modified architectures of the target instruction set processor. CAnSO can substantially reduce the design or optimization time while preserves a reasonable accuracy.

ACKNOWLEDGEMENT

This research was supported in part by Core Research for Evolutional Science and Technology (CREST) of Japan Science and Technology Corporation (JST).

REFERENCES

[1] F. Barat, R. Lauwereins, and G. Deconinck, "Reconfigurable instruction set processors from a hardware/software perspective," IEEE Trans. on Software Engineering, vol. 28, no. 9, 2002, pp. 847-861.

[2] Mibench, www.eecs.umich.edu/mibench.

[3] S. Mohanty, V.K. Prasanna, S. Neema and J. Davis, "Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation, LCTES'02-SCOPES'02, 2002, pp. 18-27.

[4] H. Noori, F. Mehdipour, K. Inoue, K. Murakami, "A reconfigurable functional unit with conditional execution for multi-exit custom instructions," IEICE Trans. ELECTRON., vol. E91-C, no. 4, April 2008.

[5] Simplescalar, www.simplescalar.com

[6] S. Yehia, N. Clark, S. Mahlke, and K. Flautner, "Exploring the design space of LUT-based transparent accelerators," Int'l Conference on Compilers, Architecture, and Synthesis for Embedded Systems, 2005, pp. 11-21.

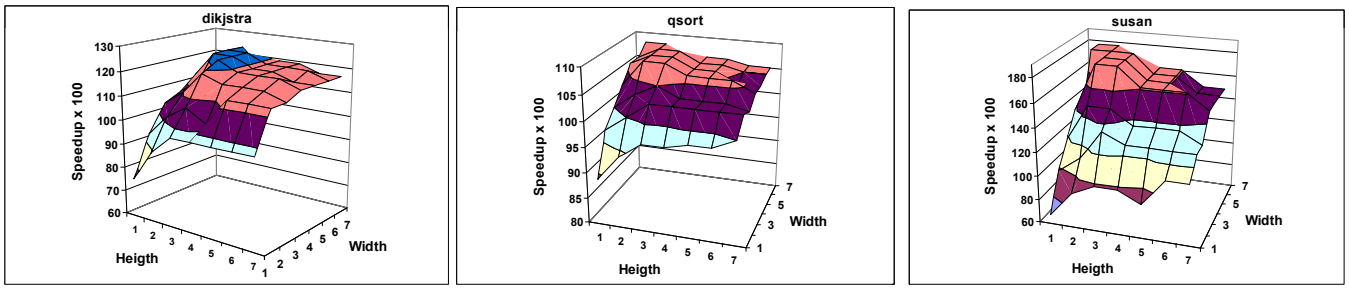

Fig. 6. Design space exploration using CAnSO for three randomly selected application