A Combined Analytical and Simulation-Based Model for Performance Evaluation of a Reconfigurable Instruction Set Processor

Mehdipour, Farhad
Faculty of Information Science and Electrical Engineering, Kyushu University

Noori, Hamid
Institute of Systems, Information Technologies and Nanotechnologies

Javadi, Bahman
Computer Engineering and IT Department, Amirkabir University of Technology

Honda, Hiroaki
Computer Engineering and IT Department, Amirkabir University of Technology

他

出版情報：Asia and South-Pacific Design Automation Conference. 14, pp.564-569, 2009-01-21
バージョン：
権利関係：

A Combined Analytical and Simulation-Based Model for Performance Evaluation of a Reconfigurable Instruction Set Processor

Farhad Mehdipour
Faculty of Information Science and Electrical Engineering, Kyushu University, Japan
farhad@c.csce.kyushu-u.ac.jp

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

Bahman Javadi
Computer Engineering and IT Department, Amirkabir University of Technology, Iran
javadi@ihpcrc.com

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

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

Abstract- Performance evaluation is a serious challenge in designing or optimizing reconfigurable instruction set processors. The conventional approaches based on synthesis and simulations are very time consuming and need a considerable design effort. 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 and provide a valuable insight. Compared to cycle-accurate simulation results, CAnSO proves almost 2% variation in the speedup measurement.

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 reconfigurable instruction set processor 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 custom instructions (CIs). CIs are the instruction set extensions which are extracted from critical portions of target applications. CIs are mapped onto the RAC and 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 a CI is encountered and then the CI is executed on the RAC at runtime. RAC acts as an accelerator and brings about higher performance [2][7][12][10].

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. Performance evaluation models are divided into to broad classes including structural models and analytical models [5]. The former one includes empirical studies based on measurements and simulations of the target system. The latter one incorporates a system (usually simplified) structure to obtain mathematically solvable models.

* CAnSO in Japanese means simple.

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. Recent research shows that the achieved performance highly depends on the applications' characteristics [3].

Experimental Setup: Throughout this research we conducted some experiments for displaying the observations and demonstrating the achievements. The experimental setup is as follows. A reconfigurable instruction set processor matching to the general template in Fig. 1 is assumed. Fourteen applications of Mibench [8] from various domains (e.g. automotive, security, consumer, network, telecommunication) are used for generating CIs (DFGs) similar to the approach in [10].To commence with the CI generation, applications are executed on Simplescalar [11] as an ISS (Instruction Set Simulator) and profiled to identify the hot segments. Then, the CIs are extracted from the hot segments (excluding floating point, control and memory access instructions except one store). The number of CIs generated for different applications varies from one (in *crc*) up to 80 (in *rijndael*). Also, CIs include at least five and at most 59 instructions. Simplescalar's cycle accurate simulator [11] has been extended to simulate a reconfigurable instruction processor. According to Fig. 2 the RAC is responsible for executing almost 30% of dynamic instructions of applications in average. The more execution on the RAC, higher speedup is achievable.

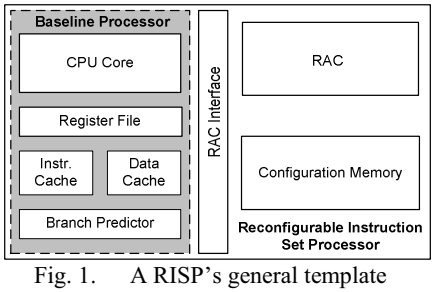

Fig. 1. A RISP's general template

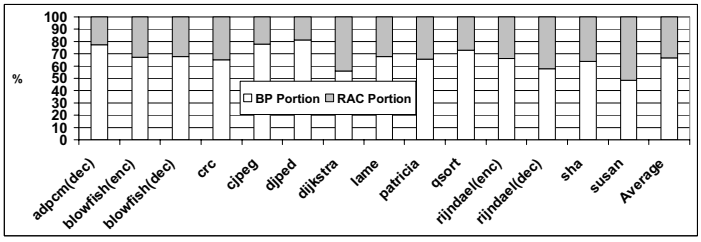

Fig. 2. Breakdown of applications' exec. time on the BP and RAC

978-1-4244-2749-9/09/$25.00 ©2009 IEEE

II. 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 [9]. Furthermore, it should be noted that some simulations are never to end. 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. The proposed model (CAnSO) has a qualitative insight in providing trace-driven information from the application. On the other hand, in the CAnSO, some of the parameters are obtained by means of cycle-accurate simulation of applications. Therefore, it emphasizes on both real observations and analytical model.

Fig. 3 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 that will be later introduced in detail) 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. 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. Processor's Template

Fig. 1 illustrates processor's structure. In our template architecture, 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. CIs are indexed for direct accessing of the associated configuration bit-stream from the configuration memory. To access required operands in the RAC, the content of all registers are sent to the RAC (by sharing the register file between RAC and conventional functional units) and the registers are used with respect to configuration inside of the RAC. Controlling configurations i.e. the task of loading and initiating the RAC, can be hardware or software-based. In a software-based mechanism starting address of a CI is replaced with a special instruction (might be referred as invocation instruction [10]). 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 a quick retrieval of the corresponding bit-stream. Memory accesses might be included or excluded. In case of inclusion, the impact of data and instruction caches hits/misses should be taken into account. In addition, in case of presence of control instructions in the CIs, the effect of branch results should be applied to the model.

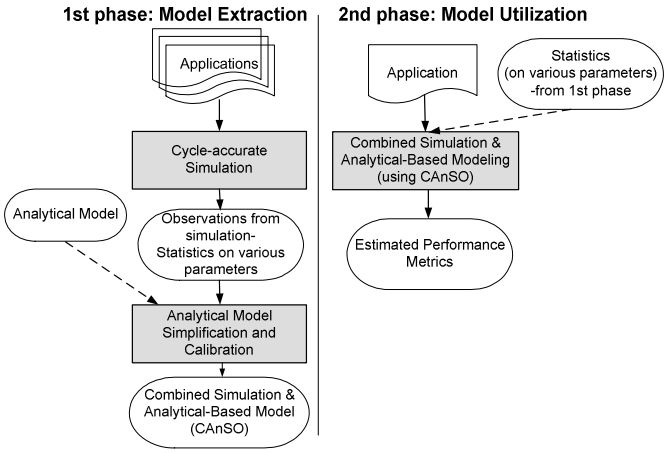

Fig. 3. Model extraction and model utilization phases

B. Basic Model Definitions

Below our terminology and some definitions are presented:

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

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

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

$\theta_i = (\theta_{i1}, \theta_{i2}, \theta_{i3}, ..., \theta_{i,m_i})$, 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 execution. Correspondingly, $O_i = \sum_{j=1}^{m_i} \theta_{ij}$ is the total number of executions of CI_i.

Two type of execution are assumed for CI_i:

Single execution: in the j^{th} occurrence of CI_i, it is executed once ($\theta_{ij} = 1$). In Fig. 4, second occurrence of CI_1 and first occurrence of CI_2 are the single executions ($\theta_{12} = 1$, $\theta_{21} = 1$). Fig. 4 also indicates that an overhead time exists in either of CI loadings on the RAC.

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. 4 shows that CI_1 and CI_2 in their first and second occurrences are executed continuously ($\theta_{11} = 2$, $\theta_{22} = 3$).

Based on the 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 $C_i = \{j \mid j \in \{1,...m_i\}, \theta_{ij} > 1\}$.

n_{CI} : The total number of CIs.

τ_{RAC} : A fixed-delay for executing a CI on the RAC. This timing factor also includes all latencies concerning to execution of a CI on the RAC.

τ_{OVH} : The overhead time including RAC reconfiguration time, the time required for setting RAC up and communicating between RAC and BP. This overhead time also depends on the

controlling mechanism of the configurations. In the software-based method,

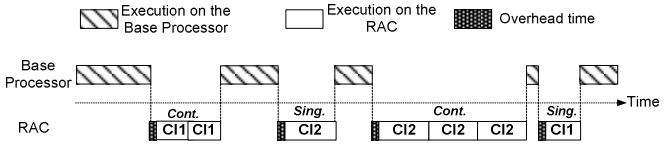

Fig. 4. Various types of a CI execution on the RAC

an additional cycle is spent for executing the CI invocation instruction whilst in the hardware-based method no additional cycle is considered.

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

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

Therefore, speedup for CI_i can be calculated as:

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

C. 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. The numerator of Eq. 3 represents the portion of application's execution time elapsed on the RAC and the numerator of Eq. 4 denotes the application execution time that is run on the BP. According to the Amdahl's law, the overall speedup (s_o) can be calculated as Eq. 5. The first term of denominator denotes the portion of execution time associated to the BP and second term of denominator represents the summation of execution times of a CI on the RAC divided by partial speedup related to that CI. The execution on the RAC and BP are not performed concurrently; therefore the total execution time of application on the reconfigurable instruction set processor is the summation of two above mentioned terms.

$$f_{RAC} = \frac{\sum_{i=1}^{n_{CI}} \left(\tau^i_{BP} \times \mathrm{O}_i \right)}{n_{tcc}} \quad (3) \qquad f_{BP} = \frac{\left(n_{cc} - \sum_{i=1}^{n_{CI}} \left(\tau^i_{BP} \times \mathrm{O}_i \right) \right)}{n_{tcc}} = 1 - f_{RAC} \quad (4)$$

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

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

D. The Effect of CI Length

In a CI generation tool as a part of compilation tool, the number of available resources (functional units) and other architectural constraints of the RAC should be considered during CI generation [10]. One important task of the compilation tool is to mapping CIs onto the RAC which is referred as mapping tool. A mapping tool takes a data flow graph of a CI and tries to map its nodes onto the functional units of the RAC [6]. 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 [6] is one solution. These CIs should be subsequently loaded and executed on the RAC.

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 temporal partitioning algorithm can be utilized for dividing a large CI to a number of smaller CIs without violation of the timing constraints. It should be noted that a large CI 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 each CI_k :

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

In Eq. 5, $\psi(\theta, \tau)$ would be replaced with $\psi'(\theta', \tau)$.

E. 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 [10]. 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 instruction caches misses might be reduced [10]. 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 cache miss rate. Eq. 8 is introduced for overall speedup (s_o) calculation and includes the side-effects as well.

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

F. RF's Input/Output Ports

Register file is shared between BP and RAC in the template architecture. It is assumed that $\Delta_{reg} / \nabla_{reg}$ are the number of read/write ports of the register file, and $\Delta^i_{CI} / \nabla^i_{CI}$ 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:

$$\tau'_{OVH} = \tau_{OVH} + \max(0, \left\lceil \frac{\Delta^i_{CI} - \Delta_{reg}}{\Delta_{reg}} \right\rceil) + \max(0, \left\lceil \frac{\nabla^i_{CI} - \nabla_{reg}}{\nabla_{reg}} \right\rceil) \quad (9)$$

G. RAC's Delay

The RAC's delay is an essential element regardless of whether the analytical or simulation approaches are used for performance evaluation. One way is to synthesizing the RAC and obtaining its delay which is time consuming particularly in case of need for examining different architectures (e.g. in a design space exploration process). Here, a simple approach is introduced which estimates the latency of the critical path in the RAC 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 the width equal to *w* and height equal to *h* and basically has a combinational logic (Fig. 5). Each FU implements an instruction level operation. RAC does not have any temporal storage unit such as local memory or register file. Multiplexers (muxes) are utilized as routing resources to route appropriate data between FUs. 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 shows a connection to the same row). A piece of connection scheme is depicted in Fig. 5.

It is assumed that all FUs in the RAC 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. Delays of FUs and muxes are achieved by synthesizing them or using the pre-synthesized library information. The delay for all FUs is similar, but, different sizes of muxes are synthesized to achieve their latencies. Consequently, critical path delay of RAC_h^w can be calculated as:

$$\tau_{RAC}{}_h^w = \sum_{i=1}^{h} \tau_{FU} + \sum_{i=1}^{h-1} \tau_{MUX}{}_i^k, \quad k \in \{0,1,...,w\} \tag{10}$$

where, $\tau_{mux}{}_j^i$ is *j*th mux between rows *i* and *i+1*.

Increasing values of *h* and *w* can affect the critical path delay of the RAC, due to their impact on the number of FUs and muxes locating in the critical path and the size of muxes as well. It is assumed that entire muxes including mux(2^n to 1) are available and other mux sizes should be replaced with the closest greater size mux. For instance, all muxes including mux(5 to 1), mux(6 to 1) and mux(7 to 1) are replaced with a mux(8 to 1). In Eq. 8, we replace $\psi'(\theta',\tau)$ with $\psi'\left(\theta',\tau_{RAC}{}_h^w + \tau_{OVH}\right)$ to reflect the effect of RAC's dimensions in the speedup evaluation. Obviously, for a different RAC architecture, a corresponding delay model should be replaced.

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, hence 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. Fig. 6 depicts 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. This ratio can be measured during the simulation in model extraction phase. Fig. 7 shows the fraction of single CIs encountered in the attempted applications as well as the fraction of CIs comprising the CIs generated from partitioning large CIs. For some applications e.g. *cjpeg* and *djpeg* the number of single executions increases after partitioning large CIs. In some applications like *crc* or *adpcmd*, CIs are executed ceaseless, thus $\alpha = 0$. On the contrary, for *patricia* or *qsort* almost all CI executions are intermittent. The average value for α is almost 43%. Putting altogether, following equations would be obtained:

$$s_o = \frac{n_{tcc}^*}{\left(n_{tcc}^* - \delta_{im}^* \times p_{im}^* - \sum_{i=1}^{n_{DI}^*}\left(\tau_{BP}^i \times O_i^*\right)\right) + \psi'\left(\theta', \tau_{RAC}{}_h^w + \tau_{OVH}\right)} \tag{11}$$

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

In above equations, all variables marked with the * are obtained via simulation in the model extraction phase (Fig. 3).

V. Experiments

Some experiments were conducted to validate CAnSO and study the significance of different variables in the performance evaluation as well (due to space limitation, only some part of results are given here). Firstly, our analytical model is established according to the first phase in Fig. 3. All attempted applications (including fourteen applications from [8]) 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. As aforementioned, Simplescalar [11] has been modified for cycle-accurate simulation of the intended processor. Other experimental setup details were mentioned in Section 1. The RAC structure comprises sixteen FUs locating in five rows including six, four, three, two and one FUs, respectively (similar to a RAC in [6][10]). The basic elements of RAC including FUs and various size multiplexers were synthesized and their delays were obtained.

Then, the delay of RAC ($\tau_{RAC}{}_{h=5}^{w=6}$) is calculated using the approach in Section III.G. Due to tight-coupling of the RAC to the BP, τ_{OVH} is assumed to be one clock cycles that is reasonable, because the configuration bit-stream size is hundreds of bits [10]. The clock frequency of the BP which is a MIPS-like RISC processor is 200MHz.

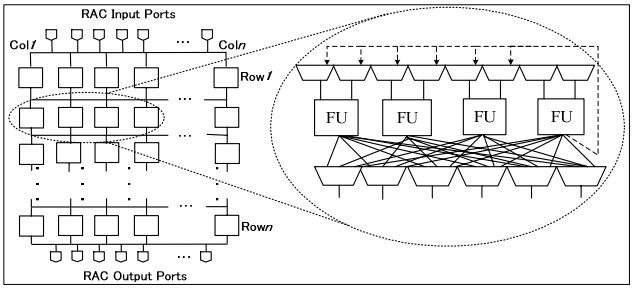

Fig. 5. Assumed RAC architecture

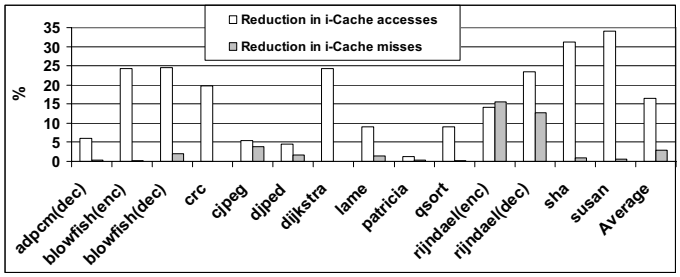

Fig. 6. Percentage of reduction in i-Cache accesses and misses

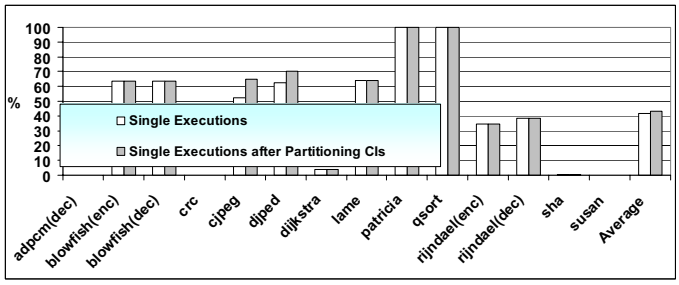

Fig. 7. Fraction of CIs including single execution

A. *Model Validation*

Fig. 8 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. 8, 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 in those applications.

B. *Design Space Exploration Using CAnSO*

Designing an appropriate RAC for a reconfigurable instruction set processor is a challenging issue. In [2], [4] and [10] a quantitative approach has been used to cope with this issue. This approach strongly depends on designer observations and decisions on the statistics gathered from the applications. That is time consuming and challenging approach that might necessitate examining some design points to specify a design gaining more performance [10].

In [3] and [12] various design parameters are examined through a design space exploration (DSE). 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 (e.g. number of functional units or processing elements, input and output ports, type of functional units and so on). One way for involvement of several parameters in the design procedure is to efficiently exploring the design space [9]. 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 17 days while using CAnSO, it reduces to the range of hours (almost four hours). More importantly, if CI generation tool would be similar to one introduced in [10] which is based on dynamic profiling (i.e. profile is achieved via application simulation), then it is possible to gather the required information for establishing the analytical model during CI generation. Consequently, re-simulating applications is not needed thus required time and efforts for extracting model are alleviated.

Fig. 9 shows the speedup variation with respect to different RAC dimensions for some applications of Mibench [8]. Increasing the width of RAC increases speedup because of taking benefits of parallelism in DFGs (of CIs), however for the widths larger than six, no more speedup is attainable due to negative impact of growing the number of routing resources and also their sizes on the RAC's delay and area. Obviously, among RAC designs with similar speedup a design with smallest area is preferred. Additionally, with respect to the height alterations three different states are observed in the speedup variation. According to Fig. 9, for the small heights the speedup is very low because CIs with longer execution sequences can not be mapped on the RAC without partitioning. On the other hand, when the height increases, the speedup and area rises as well. For the higher heights (almost more than five) the speedup declines again due to the RAC's longer delay. It should be noted that for a different CIs set and RAC architecture the above mentioned behavior might be different.

Another important advantage of CAnSO is to amortizing the time required for repeating the simulation when some modifications are applied to the design. Each iteration of the CAnSO takes less than a minute, even if modifications are made in the RAC's architecture. To show the capability of CAnSO in case of alteration of architectural specification, the speedup for different number of read/write ports of RF have been obtained from simulation and CAnSO as well. Increasing the number of read/write ports (regardless of its negative effect on the area and energy consumption) brings about more speedup. Aside from that, Fig. 10 reveals that CAnSO produces results similar to cycle-accurate simulation results. In the majority of CIs, the number of inputs and outputs are more than 2 and 1, respectively. Therefore, extra clock cycles are required when an RF with 2-read/1-write port is used which reduces the achievable speedup. On the other hand, speedup obtained for 4-read/2-write port RF and 8-read/4-write one are almost similar.

VI. Conclusion and Future Work

An analytical model for speedup evaluation of a reconfigurable instruction set processor was proposed. To become more accurate and realistic, this model is established and

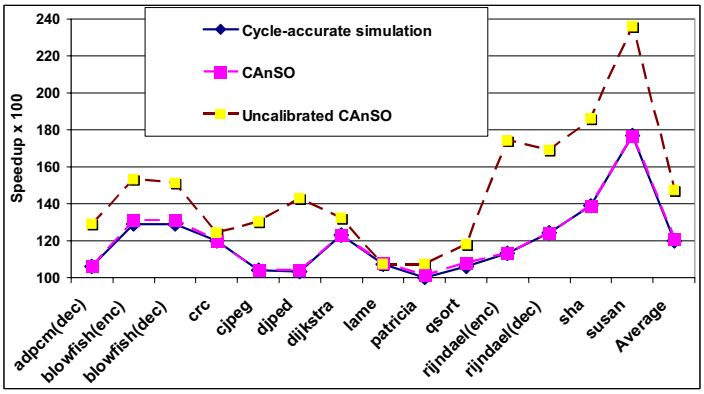

Fig. 8. CAnSO successfully tracks the cycle-accurate simulation while uncalibrated CAnSO results in 22% difference in average

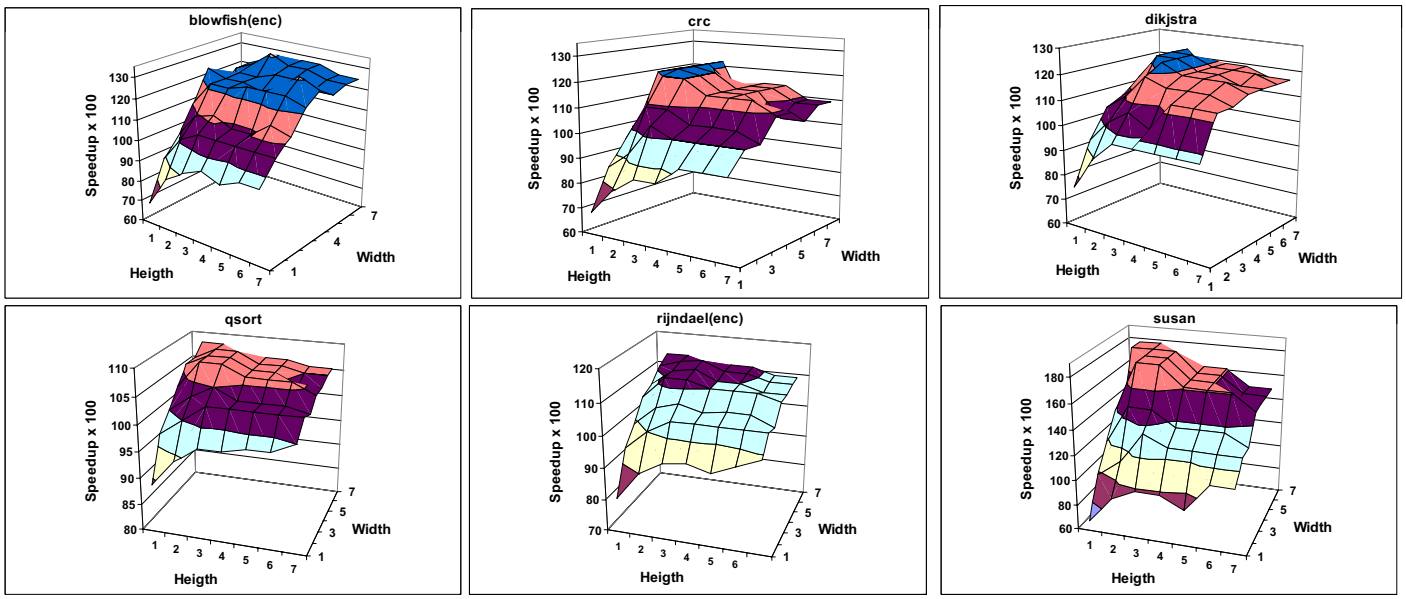

Fig. 9. Design space exploration using CAnSO for some applications

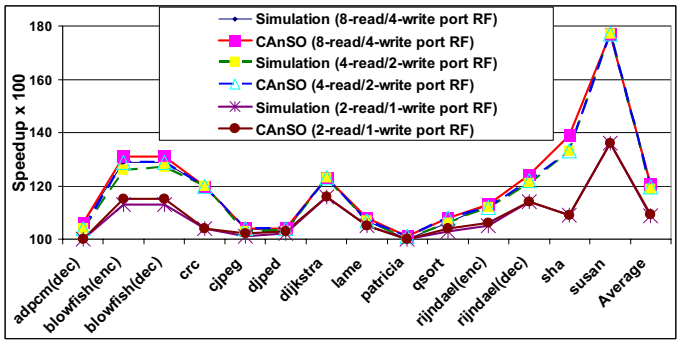

Fig. 10. CAnSO generates reliable results in a short time when design parameters change

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. That is also suitable when a large number of designs should be examined in an exploration procedure of a large design space. CAnSO can substantially reduce the design or optimization time while preserves a reasonable accuracy because of construction based on the realistic information of cycle-accurate simulations. It proves less than 2% variation in evaluation results while uncalibrated CAnSO depicts 22% difference in average. Moreover, it can be expanded to handle other design metrics (e.g. energy consumption) as well. The proposed model is a naive and starting in the domain of reconfigurable instruction set processors. In the future, we intend to expand our model to support CIs which cross loop boundaries and include control instructions. In addition, more complicated RAC architectures will be considered.

Acknowledgments

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

References

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

[2] Clark, N., Kudlur, M., Park, H., Mahlke, S. and Flautner, K., "Application-specific processing on a general-purpose core via transparent instruction set customization," *The 37th Symp. on Microarchitecture*, pp. 30-40, 2004.

[3] Enzler, R. and Platzner, M., "Application-driven design of dynamically reconfigurable processors," *http://e-collection.ethbib.ethz.ch/browse/sg/092_e.html*, 2001.

[4] Kim, Y., Kiemb, M. and Choi, K., "Efficient design space exploration for domain-specific optimization of coarse-grained reconfigurable architecture," *In Proc. of SoC Design Conference*, pp. 12-17, 2005.

[5] Kumar, B. and Davidson, E.S., "Computer system design using a hierarchical approach to performance evaluation," *Communications of the ACM*, Vol. 23, No. 9, 1980.

[6] Mehdipour, F., Noori, H., Saheb Zamani, M., Inoue, K. and Murakami, K., "Design space exploration for a coarse grain accelerator," *Proc. of* 13th *Asia and South-Pacific Design Automation Conference (ASPDAC)*, Korea, 2008, pp. 685-690.

[7] Mei, B., Vernalde, S., Verkest, D., Lauwereins, R., "Design methodology for a tightly-coupled VLIW/reconfigurable matrix architecture: A Case Study," *Design, Automation and Test in Europe*, Vol. 2, pp. 21-24, 2004.

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

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

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

[11] Simplescalar, www.simplescalar.com

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