Automated Identification of Performance Bottleneck on Embedded Systems for Design Space Exploration

Yuki Ando, Seiya Shibata, Shinya Honda, Hiroyuki Tomiyama, Hiroaki Takada

Automated Identification of Performance Bottleneck on Embedded Systems for Design Space Exploration

Yuki Ando[1], Seiya Shibata[1] *, Shinya Honda[1], Hiroyuki Tomiyama[2], and Hiroaki Takada[1]

[1] Nagoya University, Nagoya, Japan
[2] Ritsumeikan University, Kusatsu, Japan

Abstract. Embedded systems usually have strict resource and performance constraints. Designers often need to improve the system design so that the system satisfies those constraints. In such case, performance bottlenecks should be identified and improved effectively. In this paper, we present a method to identify performance bottlenecks. Our method automatically identifies not only the bottlenecks but also a list of improvement rates of bottlenecks that is necessary for the system to satisfy design constraints. With the list of improvement rates, designers easily consider how to improve the bottlenecks. A case study on AES encryption and decryption application shows effectiveness of our method.

1 Introduction

As the functionality of embedded systems has increased, they are required more and more computation. Some processes representing the functions of the systems are implemented in dedicated hardware so that the system accelerates their computation. On the other hand, implementing processes in dedicated hardware causes to increase hardware area. For embedded systems, it is important to satisfy design constraints such as execution time and hardware area. So system designers are facing a problem that they have to efficiently find a system configuration satisfying the design constraints.

Figure 1 shows an example of design flow for embedded systems with dedicated hardware. It starts from changing software description to software/hardware (SW/HW) mixed description. Then, designers conduct exploration of SW/HW partitioning. During the exploration of SW/HW partitioning, system designers try to find a mapping that satisfies design constraints by changing the allocation of processes. In the past, design tools have been developed in order to efficiently explore SW/HW partitioning [1, 2].

After the exploration of SW/HW partitioning, the designers have to check whether a mapping satisfies all design constraints because exploration of SW/HW partitioning may not find a mapping that satisfies the design constraints. For

* Presently with NEC Corporation

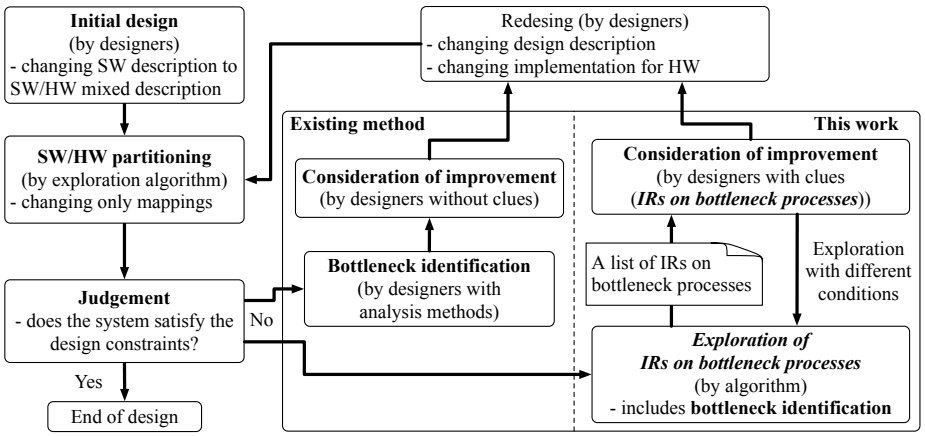

Fig. 1. Entire design flow

example, a mapping satisfies design constraint of hardware area but may not satisfy that of execution time. From this mapping, changing an allocation of a process from SW to HW makes execution time faster. However, it brings bigger hardware area. As the result, new mapping satisfies design constraint of execution time but may not satisfy that of hardware area. Thus, exploration of SW/HW partitioning does not always find a mapping that satisfies all design constraints. In such case, designers need improve the design description.

There are two big problems to improve the design description. First problem is identification of bottlenecks on the system. In the existing design method, designers identify the bottlenecks using analysis tools. Fei et al. divided execution logs into particular behavior groups and analyze the behavior groups[3]. Valle et al. proposed a method to make logging of system performance easy[4]. Second problem is identification of improvement rates (IRs) of bottlenecks. With existing design methods, designers have to identify how much they have to improve bottlenecks to satisfy design constraints. Then, they consider how to change design description to improve the system performances. The existing design methods waste time of designers to improve design description.

In this paper, we propose a method to automatically identify not only bottlenecks but also a list of IRs of bottlenecks that are necessary to satisfy the design constraints. With our method, designers no longer identify how much they have to improve the bottlenecks because our method automatically takes care of that. It is ideal for designers to know the essential IRs on bottlenecks. In addition, our method lists up several candidates to improve the systems. Thus, our method brings shorter time to identify the IRs of bottlenecks, and designers can take more time to consider various ways to improve the system description.

Our main contribution is a method to explore the IRs on bottlenecks. In addition, our method explores IRs for not only execution time but also hardware area. A case study on AES shows the effectiveness of our method.

2 Application model and target architecture

Fig.2 shows an example of application (AES encryption and decryption) model on left side. Application model describes a set of processes running concurrently and channels representing communications among processes. This kind of models is common for design tools such as ARTEMIS[5] and Metropolis[6].

Right side of Fig.2 depicts an example of our target architectures. It also shows an example of mappings (top and EncF are allocated to SW, and the others are allocated to HW) for AES encryption and decryption model. The figure shows typical architecture and mapping of system-on-a-chip. The processors (CPU) are assumed to be homogeneous, and the number of processors must be greater than or equal to one. Processors, dedicated hardware, and shared memory are connected through a standard on-chip bus. The processes allocated to SW are implemented onto processors as RTOS tasks. Those allocated to HW are implemented onto hardware modules (HWMs) in the dedicated hardware.

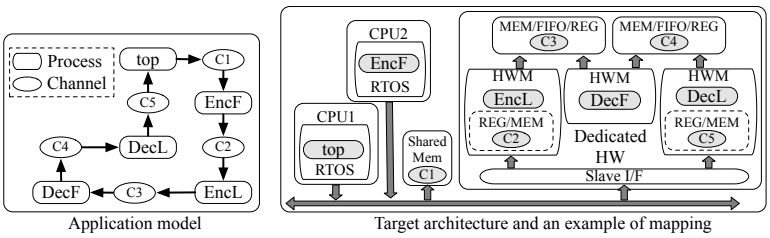

Fig. 2. An example of application model (AES) and target architecture

Memory is shared for communication among the processors (C1). For communication between the processors and hardware modules, memory (MEM) and the register (REG) are generated in HWMs (C2, C5). They are accessed from processors through a standard on-chip bus and the slave interface. HWMs can communicate with each other directly through the exclusionary FIFO, MEM, and REG (C3, C4). Thus, the architecture allows processors and hardware modules to communicate directly through a bus, memory, and the interfaces.

3 Exploration of improvement rate on bottleneck process

3.1 Definition of bottleneck process

In this paper, a process X is defined as a bottleneck process if reducing the execution time of process X shortens entire execution time of system without any change of mapping.

Figure 3 shows an example of bottleneck processes. The example has four processes. The original execution time of processes A, B, C, and D are 300, 400, 100, and 700, respectively. Processes A, B, C are mapped to a processor

(CPU) and process D is mapped to hardware module (HW). The original entire execution time is 800 as shown on the right side of the figure.

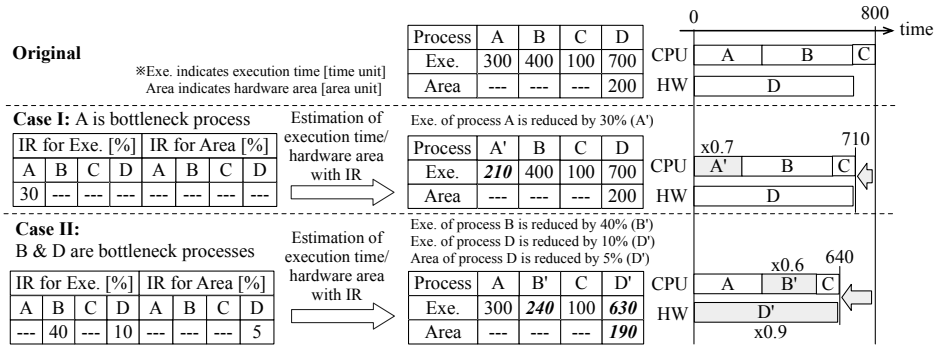

Fig. 3. An example of IRs and bottleneck processes.

Case I shows that process A is a bottleneck process. The execution time of A is assumed to be 210 (A'), that is 70% of process A. The entire execution time of Case I is reduced to 710 because execution time of A' is applied. Reducing the execution time of process A causes to shorten the entire execution time. Therefore, process A is a bottleneck process under our definition.

With our definition, several processes may become bottleneck processes at the same time as shown Case II. The entire execution time also becomes shorter than original one when the execution time of process B and D are assumed to be 240 (B') and 630 (D'), respectively. Thus, processes B and D are bottlenecks.

3.2 Definition of improvement rate (IR)

IR indicates the ratio to shorten the execution time or to reduce the hardware area of process compared to original one. Examples of IR are also shown in Fig. 3. Each process has two types of IR, one for execution time and the other for hardware area. For example, original execution time and hardware area of process D are 700 and 200, respectively. In Case II, it has 10% of IR for execution time and 5% of IR for hardware area. Thus, execution time and hardware area of process D are assumed to be 630 and 190, respectively.

3.3 Exploration of the IRs on bottleneck processes

Under the definition of a bottleneck process and that of IR, estimating the entire execution time with IRs identifies bottleneck processes. If the entire execution time is reduced, processes that have IRs are bottlenecks. Thus, increasing the value of IRs unveils whether the processes are bottlenecks or not.

Fig. 4 shows the exploration flow of IRs on bottleneck processes. The inputs are a mapping to explore the IRs and design constraints of execution time and hardware area. The output is a set of IRs that satisfies the design constraints.

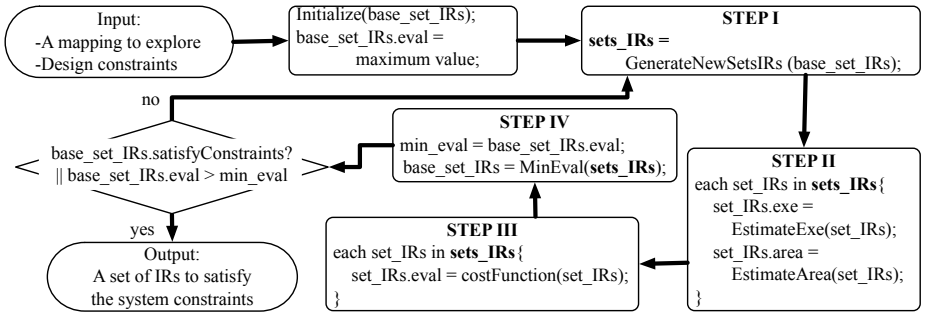

Fig. 4. Exploration flow of IRs for bottleneck processes

At the beginning, all IRs in a set of IRs (base_set_IRs) are initialized to 0% (Initialize). At the same time, evaluation value of base_set_IRs (base_set_IRs.eval) is initialized to maximum value. After the initialization, four steps are repeated. At step I, new sets of IRs are generated from base_set_IRs to increase values of IRs so that the input mapping satisfies the design constraints. At step II, execution time and hardware area are estimated with generated sets of IRs. At step III, all generated sets of IRs are evaluated by the cost function because increasing the same value of IR causes to produce unrealizable IR such as 100%. At step IV, the best set of IRs is selected for further exploration. The detail of each step is described end of this section. After step IV, if a set of IRs satisfies the design constraints, or there is no better set of IRs, the exploration ends.

After an exploration, designers get a mapping and its best set of IRs on bottleneck processes. The best set of IRs indicates how much bottleneck processes should be improved to satisfy the design constraints for the mapping. In addition, exploration on different mappings may bring better ones. Thus, designers can easily find the best mapping and its set of IRs on bottleneck processes among several pairs of them.

STEP I From base_set_IRs, new sets of IRs are generated by a function GenerateNewSetsIRs. Only an IR in base_set_IRs is increased at once. Before the exploration, designers have to define static increasing number. Generating new sets of IRs on all IRs, the number of new sets is twice the number of processes in maximum.

STEP II Execution time and hardware area are estimated for all sets of IRs in sets_IRs. Trace-based estimation tools [7, 8] are assumed to be used to estimate the entire execution time with IRs. The tools usually take profiles of process

execution time as input. For that, the tools can estimate the entire execution time with IRs by arranging the profiles of execution time. Hardware area is estimated by summation of the area of hardware modules. Area of hardware module is reduced when the hardware area is estimated with IRs.

STEP III Sets of IRs are evaluated by the cost function (costFunction) to determine the best set of IRs. Better set of IR is assumed to have smaller value. Without this evaluation, only an IR of a process may be increased. This causes to produce unrealizable value of IR such as 100%. The definition of the cost function is described in Sect. 3.4.

STEP IV A function, MinEval, returns a set of IRs that has minimum evaluated value. Because better set of IRs has smaller value, this step selects the best set of IRs among the generated sets of IRs. The selected set of IRs become base_set_IRs for further exploration.

3.4 Detail of cost function

In order to determine better sets of IRs, we propose a cost function (costFunction). First of all, the set of IRs should make a mapping satisfy the design constraints because this is the main purpose. So the cost function allows IRs to increase their values. Secondly, the set of IRs should not have impossible values of IRs such as 100%. So the cost function have to prevent that a set of IRs includes such impossible values. Thirdly, there may be some processes that are no longer improved. Such processes should not be listed to improve more. From these points, the cost function should have three features below. Note that smaller value is assumed to be better.

1. The value of cost function gets smaller (better) when the estimated execution time and hardware area close to the design constraints (the values of IRs get larger).
2. The value of cost function should be larger for large IRs to prevent impossible values of IRs.
3. Designers can set easiness of improvement on all processes separately.

For first feature, distance (*dis*) between estimated values and design constraints is used. For second feature, penalty (*penal*) depending on IRs is used. Third feature is handled by introducing values of easiness for processes.

There are three kind of parameters determined by designers before the exploration starts. Note that xxx should be either "exe" or "area" indicating execution time or hardware area, respectively.

- $target_{xxx}$: value of design constraints for exe/area
- $ease_{p\_xxx}$: value of easiness to improve process p for exe/area
- max_{xxx} : maximum value of exe/area

The inputs of the cost function are estimated values of execution time (est_{exe}) and hardware area (est_{area}), and a set of IRs ($rate_{p\_xxx}$). The cost function consists of distance (dis) and penalty ($penal$). It returns its value ($eval$) by (1).

$$eval = dis + penal \tag{1}$$

From (2) to (4) show calculation of dis. Because our method deals with execution time and hardware area, distances for each design constraints (dis_{exe} and dis_{area}) are calculated as shown in (3) and (4).

$$dis = (dis_{exe} + dis_{area}) \,/\, 2 \tag{2}$$

$$dis_{xxx} = \begin{cases} 3 \times dif_{xxx} + 0.1 & (dif_{xxx} > 0) \\ 0.001 \times dif_{xxx} & (others) \end{cases} \tag{3}$$

$$dif_{xxx} = est_{xxx} - target_{xxx} \tag{4}$$

The total penalty ($penal$) is given by (5). It is an average of penalties for execution time($penal_{exe}$) and hardware area ($penal_{area}$) given by (6). In the equation, $rate_{p\_exe}$ and $rate_{p\_area}$ indicate the IR of process $p \in P$ for execution time and hardware area, respectively. Note that P is a set of processes in the system. Standard value of easiness to improve process p ($ease_{p\_xxx}$) is assumed to be one. If it is bigger than one, it means that the process p is hard to improve, and vice versa. Values of easiness to improve process have to be determined by designers before the exploration starts.

$$penal = (penal_{exe} + penal_{area}) \,/\, 2 \tag{5}$$

$$penal_{xxx} = (\sum_{p \in P} (rate_{p\_xxx} * ease_{p\_xxx})^3) \,/\, |P| \tag{6}$$

The coefficients in the cost function are calibrated with MPEG-4 decoder application that consists of 11 processes. We calibrated them on various combinations of design constraints, mappings and number of processors. Note that the easiness of improvement was set to one during the calibration.

4 A Case study

This section shows a case study on AES encryption and decryption application (AES) in CHStone[9]. Before this study, we decided the aim to reduce hardware area while the execution time keeps the same. During the study, we improved the performance of AES twice along with the design flow shown in Fig. 1. The target architecture is Altera Stratix II FPGA [11] board which has a single soft core processor and dedicated hardware. Design constraints are execution time and hardware area (the number of look-up table (#LUTs)).

4.1 Initial design of AES encryption and decryption

AES is written in C and the number of lines is 716. We divided AES description into five processes as shown in left side of Fig. 2 so that we can use our

system-level design tool [10] and explore SW/HW partitioning. We just changed SW description to SW/HW mixed description. This design without any optimization is called initial design. In detail, the global arrays in C description are changed to shared-memory communication. Because the process named "top" is the sequencer of the application, it is not a target of improvement and exploration of SW/HW partitioning. EncF and EncL are the first half and last half of encryption, respectively. Also, DecF and DecL are the first half and last half of decryption, respectively. AES repeats encryption and decryption 10 blocks of data consisting of 16 integers for 100 times.

4.2 Improvement of initial design

We first explored SW/HW partitioning for initial design. As AES has four processes that can be allocated to SW and HW, so there are 16 mappings in total. We explored SW/HW partitioning by implementing all mappings onto the FPGA board with our tool. From this, we found 11 mappings that construct the trade-offs between execution time and hardware area. We also found that the shortest execution time and #LUTs were 1.31 seconds and 19,244, respectively.

In order to improve the design, we decided design constraints of 1.3 seconds for execution time and 18,000 for hardware area. For all processes, the easiness to improve process was set to one (default). We explored IRs for 11 mappings on trade-off with the design constraints. In this work, we used our own trace-based estimation tool [12] for exploration of IRs on bottleneck processes.

After the exploration, we had 11 sets of IRs (a set of IRs for each mapping). The best three results in terms of value calculated by the cost function are shown in Table 1. In the table, "Mapping", "IR for Exe." and "IR for Area" indicate allocation of processes, IRs for execution time and that for hardware area, respectively.

Table 1. Identified bottlenecks and IRs for initial design

ID	Mapping				IR for Exe. [%]]				IR for Area [%]			
	EncF	EncL	DecF	DecL	EncF	EncL	DecF	DecL	EncF	EncL	DecF	DecL
No.1	SW	HW	HW	HW	—	—	—	5	—	5	—	5
No.2	HW	SW	HW	25	—	15	5	—	—	—	—	
No.3	SW	SW	HW	—	40	15	5	—	—	—	—	

From the result of No.1, DecL was identified as a bottleneck for execution time. Its execution time had to be reduced 5% in order to satisfy the design constraints. So we considered how to improve its execution time with the result. Then, we decided to tune the design description because initial design was just changed to SW/HW mixed description from SW description. At redesign step, we reduced the number of memory accesses in DecL. We also unrolled the loop instructions in DecL because it was implemented on HW. After the redesign, we again explored SW/HW partitioning and found that mapping (No.1) has the

execution time of 0.83 seconds and the hardware area of 16,573 in #LUTs. Thus, we could have a design that satisfied the design constraints.

4.3 Summary of case study to improvement AES application

As mentioned before, we improved AES twice along with the design flow shown in Fig. 1. The previous section shows the detail of first improvement of AES. In this section, we summarize our case study.

Table 2. Execution time (Exe.) and hardware area (Area) for designs

	Allocation of process / Difference from Initial †				Measured	
Design	EncF	EncL	DecF	DecL	Exe. [sec]	Area [#LUTs]
Initial	HW	HW	HW	HW	1.31	19,244
Imp1	SW	HW	HW	HW / M, L	0.83	16,573
Imp2	HW / S	HW / S	HW	SW / M	1.29	12,501

†M: reducing memory accesses, L: loops are unrolled,S: reducing hardware area

Table 2 shows execution time and hardware area of three designs. Initial, Imp1 and Imp2 indicate initial design, first improved design and second improved design, respectively. After first improvement, we got a design named Imp1 that satisfied original design constraints (shorter than 1.3 seconds and less than 18,000 in #LUTs). The aim to improve AES was reducing hardware area while the execution time remains less than 1.3 seconds. As the execution time of Imp1 was 0.83 seconds, so we considered that sacrificing the execution time could reduce the hardware area.

We, once again, decided the design constraints of 1.3 seconds for execution time and 14,000 for hardware area. Then we explored SW/HW partitioning on Imp1. We, however, could not find any mapping satisfying new design constraints. Thus, we explored IRs on Imp1 design. With the results of exploration for IRs, we changed the synthesis option of hardware for processes EncF and EncL. The synthesis options are shown on Imp2 in Table 2. As a result, we got a design whose execution time and hardware area are 1.29 seconds and 12,501 in #LUTs that satisfied the design constraints.

This case study shows that we could improve AES by using only explored IRs. After the twice of improvements, hardware area was reduced 35% with shorter execution time compared to initial design. Therefore, IRs helped the designer to tune the description of application to satisfy design constraints.

5 Conclusion

We proposed a method to identify system bottlenecks and explore improvement rates of them for embedded systems. Because our method automatically identifies not only bottlenecks but also a list of improvement rates that is necessary to

satisfy the design constraints, our method helps designers to improve the system without a time-consuming analysis. The case study on AES encryption and decryption application showed that our method surely identified system bottlenecks automatically. The designers efficiently consider how to improve the system with the list of improvement rates. Entire design time is shortened with our method. Therefore, our method is effective to improve the embedded systems.

Acknowledgment This work was in part supported by STARC (Semiconductor Technology Academic Research Center).

References

1. Dömer, R., Gerstlauer, A., Peng, J., Shin, D., Cai, L., Yu, H., Abdi, S., Gajski, D.D.: System-on-Chip Environment: A SpecC-Based Framework for Heterogeneous MPSoC Design. EURASIP Journal on Embedded Systems. (2008)
2. Ha, S., Kim, S., Lee, C., Yi, Y., Kwon, S., Joo, Y.P.: PeaCE: A Hardware-Software Codesign Environment for Multimedia Embedded Systems. ACM Trans. Design Automation of Electronic Systems. (2007).
3. Gao, F., Sair, S.: Long term Performance Bottleneck Analysis and Prediction. International Conference on Computer Design. pp. 3–9 (2006)
4. Del Valle, P.G., Atienza, D. , Magan, I., Flores, J.G., Perez, E.A., Mendias, J.M., Benini, L., Micheli, G.D.: A complete multi-processor system-on-chip FPGA-based emulation framework. International Conference on Very Large Scale Integration. pp. 140–145 (2006)
5. Pimentel, A.D.: The Artemis workbench for system-level performance evaluation of embedded systems. International Journal of Embedded Systems. 3, 181–196 (2008)
6. Balarin, F., Watanabe, Y., Hsieh, H., Lavagno, L., Passerone, C., SangiovanniVincentelli, A.: Metoropolis: An Integrated Electronic System Design Environment. Computer. (2003)
7. Ueda, K., Sakanushi, K., Takeuchi, K., Imai, M.: Architecture-level performance estimation method based on system-level profiling. In: Computers & Digital Techniques, pp. 12-19 (2005)
8. Wild, T., Herkersdorf, A., Lee, G. Y.: TAPES - Trace-based architecture performance evaluation with SystemC. Design Automation for Embedded Systems. 10, 157–179 (2005)
9. Hara, Y., Tomiyama, H., Honda, S., Takada, H.: Proposal and Quantitative Analysis of the CHStone Benchmark Program Suite for Practical C-based High-level Synthesis. Journal of Information Processing. 17, 242–254 (2009)
10. Shibata, S., Honda, S., Tomiyama, H., Takada, H.: Advanced SystemBuilder: A Tool Set for Multiprocessor Design Space Exploration. In: International SoC Design Conference, pp. 79–82. (2010)
11. Altera Corporation, `http://www.altera.com/`
12. Shibata, S., Ando, Y., Honda, S., Tomiyama, H., Takada, H.: A Fast Performance Estimation Framework for System-Level Design Space Exploration. IPSJ Transactions on System LSI Design Methodology. 5, 44–55 (2012)