Architecture-level Performance Estimation for IP-based Embedded Systems

Kyoko UEDA, Keishi SAKANUSHI, Yoshinori TAKEUCHI and Masaharu IMAI
Graduate School of Information Science and Technology, Osaka University

Abstract

In this paper, we propose a architecture-level performance estimation method for IP-based embedded systems using system-level profiling. Our method enables the performance estimation by the following procedures; 1) System-level profiling. 2) Automatic construction of the execution dependency graph (EDG) from the profile information. 3) Estimation of the system performance based on the EDG analysis. Our method enables fast performance estimation because it can estimate the performance of various architectures from the same system-level profile information. Experimental results show that our estimation method is about 10,000 times faster than the architecture-level simulations.

1. Introduction

In recent years, IP-based design methodology that aims to reduce the design time by reusing modules designed in past, called IP (Intellectual Property), has been proposed[2].

Embedded systems operating multimedia or telecommunication applications have strict design constraints relating to performance and hardware area, and there is the relation in the trade-off between the performance and the area.

Because the architecture of an embedded system affects the design quality of the system, designers should evaluate the performance of architectures consisting of different IPs and different bus protocols, in order to find out which architectures satisfy the design constraints. Problems arise when using architecture-level simulations[1, 4] because they take a long time.

Some models[3, 6, 7] are proposed to estimate system performance much faster, but it is difficult to evaluate many architectures with different bus topologies and module implementations in the limited design term, because each target system must be modeled on these models. Therefore, it takes a very long time to compare the estimated performance of the architectures and to select best architectures that satisfy demand.

This paper proposes an architecture-level performance estimation method using system-level profiling for architectures consisting of different bus topologies and module implementations. Our method mainly targets the multimedia or telecommunication systems that are data flow oriented and communication centric and periodic systems. Because execution dependencies in system-level between the data processings and transfers do not depend on the target architectures, the system performance on a range of architectures can be estimated from the same system-level profile information. The key issues of our method are as follows; 1) Focusing on the system-level profiling that obtain execution dependencies between data processings and transfers. 2) Estimation of the performance of various architectures from the only one time profile information. 3) Fast evaluation of the performance based on the analysis of the execution dependency graph.

The rest of this paper is organized as follows. Section 2 surveys the related work of the performance estimation. Section 3 describes the concept of our method and section 4 describes its detail. The experimental results in section 5 show the advantages of our method. Finally we draw our conclusion and outline our plans for future research.

2. Related work

Because the performance of the system varies according to the architecture, the designers need the system performance evaluation of a number of architectures in order to find out an optimum one that satisfies design constraints and has smaller area and higher performance.

Architecture-level simulation[1, 4] of the target system is mostly used to estimate system performance. Because it simulates complete behavior of the target system, one time simulation takes very long time. Hence, some models such as the simulation model for bus transfers and the model of real-time operating systems are proposed[3, 6, 7]. These models need the data transfer timing information extracted from a simulation result or designer's guess and each target system must be modeled on these models. Therefore, it takes a very long time to compare the estimated performance of the architectures with different bus topologies and module implementations and to select best architectures that satisfy demand in the limited design term.

Our method supposes IP database [5] storing the execution time of each data processing of modules, and esti-

1530-1591/04 $20.00 (c) 2004 IEEE

mates the performance on the target bus topology and module implementation. We focus on the dependencies between the data processings and transfers that are independent of the architecture. In our estimation method, the timing of the data processings and transfers are obtained from system-level profiling, and then, the system-level execution dependency graph (SL-EDG) representing dependencies at the system-level between the data processings and transfers is constructed. Subsequently, the architecture-level execution dependency graph (AL-EDG) is constructed and analyzed to estimate the performance of the system at the target architecture. We use SystemC language[8] for system-level profiling because SystemC allows designers to implement original channels. The next section outlines the concept of our method.

3. System-level profiling for performance estimation

Our method has the following estimation flow;

1. System-level profiling. The profile information contains the timing of the data processings and transfers. We use SystemC language and implement the monitoring channel for system-level profiling.
2. Construction of the system-level execution dependency graph (SL-EDG) from the profile information. Each vertex of the SL-EDG represents a data processing of the process or a data transfer through a channel. The data transfer vertex contains a data size equal to the data size of the channel. Edges of the SL-EDG represent dependencies on system-level.
3. Construction of the architecture-level execution dependency graph (AL-EDG) from the architecture-level model and the SL-EDG. The AL-EDG consists of all vertices and edges of the SL-EDG and the edges representing dependencies on architecture-level.
4. Estimation of the architecture performance based on the AL-EDG analysis.

Even if the designer estimates the performance of various architectures, system-level profiling and SL-EDG construction need not be undertaken for each target architecture. Therefore, it is possible to estimate the performance of various architectures from one profile information. In the following section, we briefly explain how to estimate the performance of various architectures from the architecture-level execution dependency graph (AL-EDG). The algorithm detail of our method is described in section 4.

3.1. System-level model

The system-level model SLM) comprises the processes performing data processings in parallel and the channels undertaking data transfers between two processes or between a process and the outside of the system. The process receives data through receiving channels, executes a data processing and sends data through sending channels. At the system-level model, each process contains no information of the execution time.

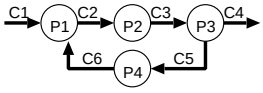

Figure 1. An example of an SLM.

There are two dependencies, **R/E Dependency** and **E/S Dependency**, between a data processing of the process and a data transfer of the channel. The **R/E Dependency** means that after data has been **received**, the **execution** of the data processing starts, and the **E/S Dependency** means that after the **execution** of the data processing, data **sending** starts.

Figure 1 shows an example of a system-level model. Pi stands for a process, and Ci stands for a channel and a direction of the data transfer.

3.2. Architecture-level model

An architecture-level model ALM) comprises the modules performing data processings and the buses undertaking data transfers. In this paper, modules and buses are assumed to have the following characteristics;

- IP modules undertake data processings of the processes of the system-level model that are mapped to the module.
- Buses undertake data transfers at the channels of the system-level model that are mapped to the bus.
- All modules and buses can execute the data processing or transfer in parallel.
- Each port of a module and each port of the system have buffers. The buffers that the module is reading cannot be overwritten by the data transfer, and the module cannot write operated data to the buffers containing untransfered data.
- The IP modules are already designed and the IP database stores the module information. The module information includes the area and the execution time of each data processing.
- The modules choose the data processing that has the highest priority from the data processings mapped to the module and executable at the same time.
- The bus chooses the data transfer that has the highest priority from the data transfers requesting to use the bus at the same time.

From these characteristics, there are two dependencies, **E/R Dependency** and **S/E Dependency**, between a data processing of the process and a data transfer of the channel. If all buffers for receiving data are occupied by data

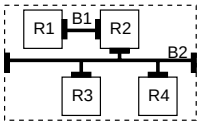

Figure 2. An example of an ALM.

that has not been processed, subsequent data cannot be **received** until a buffer becomes available by the completion of the data processing **execution**. This dependency between data processing **execution** and a data **reception** is called the **E/R Dependency**. In a similar way, if all buffers for sending are occupied by data that has not been sent to destination modules, the **execution** of the data processing for next data waits until a buffer becomes available after data **sending**. This dependency between a data **sending** and processing **execution** is called the **S/E Dependency**.

Figure 2 shows an example of an architecture-level model. Ri stands for a module and Bi stands for a bus. Bus B2 connects module R2, module R3, module R4 and the outside of the system.

3.3. Performance estimation using execution dependency graph

This section explains how to estimate system performance using architecture-level execution dependency graph (AL-EDG).

In this section, we explain the AL-EDG analysis flow with two examples of the architecture-level model of system S1 that has process P1 and channel C1 and C2. Process P1 receives data from channel C1, operates the received data, and sends the operated data through channel C2.

- Architecture A1 consists of module R1 to which process P1 is mapped and bus B1 to which channel C1 and channel C2 are mapped. Channel C1 has higher priority than channel C2. Each port of architecture A1 has one buffer.
- Architecture A2 consists of module R2, bus B21 and bus B22 to which process P1, channel C1 and channel C2 are mapped, respectively. Each port of architecture A2 has one buffer.

Figure 3 shows the AL-EDG of system S1. The vertex stands for the data transfer and processing, and the edge represents a dependency between them.

According to the **R/E Dependency**, it is possible to execute the data processing of process P1 after the data transfer of channel C1, thus, edge RE_i, representing the i-th **R/E Dependency** described in section 3.1, exists between vertex $C1_i$ and vertex $P1_i$. As the same manner, edge ES_i representing the **E/S Dependency** exists.

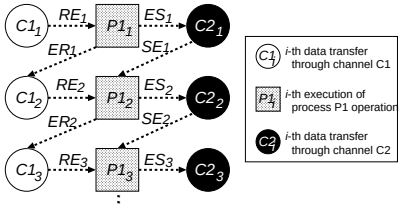

Figure 3. The AL-EDG of system S1.

Considering the target architecture, the **E/R Dependency** and **S/E Dependency** described in section 3.2 exist between the data processing of the process to which is mapped a module and the data transfer of the channel to which is mapped a bus. Edge ER_i and edge SE_i in Figure 3 represent the **E/R Dependency** and **S/E Dependency**, respectively.

System S1 performs as follows in architecture A1. At first, system S1 initiates the data transfer of vertex $C1_1$ that has no dependency from other vertices, then operates the data processing of vertex $P1_1$ that has a dependency from vertex $C1_1$ starts. After the completion of the data processing of vertex $P1_1$, the data transfers of vertex $C1_2$ and vertex $C2_1$ are able to be executed, and the data transfer of vertex $C1_2$ starts because channel C1 has higher priority than channel C2. Because vertex $P1_2$ has dependencies not only from vertex $C1_1$ but also vertex $C2_1$, the data transfer of vertex $C2_1$ starts after the completion of the data transfer of vertex $C1_2$, and the data processing of vertex $P1_2$ starts after the completion of the data transfer of vertex $C2_1$.

In the same way, system S1 performance in architecture A2 can be analyzed. Channel C1 and channel C2 is mapped to the different buses, then the data transfers at channel C1 and channel C2 can operate in parallel.

If the execution time of all AL-EDG vertices are given, the finish time of each vertex operation can be estimated by adding the execution time to the starting time of the data processing or transfer. Thus, the finish time of the operation of the vertex that is analyzed at the end represents the execution time of the system.

To construct the AL-EDG for the target architecture, the same SL-EDG can be used because the SL-EDG has no edges related to the target architecture. Therefore, the SL-EDG is constructed only once. Section 4.2 describes the SL-EDG construction method in detail.

3.4. Process execution time and data transfer time

It is necessary to obtain the execution time of each vertex for the performance estimation based on the AL-EDG analysis. The execution time of a process vertex is equal to the execution time of the module to which the process is mapped. This is stored in the IP database. The execu-

tion time of the channel vertex changes with the bus protocol. Our method can estimate any bus protocol because the execution time of the channel vertex is assumed to be calculated from such variables as the data size, the bit-width and the rate of the bus that the channel is mapped to for target bus protocol.

The data transfer size is equal to the channel data size obtained from system-level profiling. Moreover, they are independent of the architecture, therefore, a profile information of one time system-level simulation can be used for the performance estimation of various architectures.

4. System performance estimation method

4.1. System-level profiling using SystemC

This section proposes the method of the system-level profiling using SystemC.

SystemC, a system-level description languages, consists of libraries of the software programming language C++, and can describe some concepts of hardware, for example, parallel executions[8]. In SystemC version 2.0, the designers can implement the data transfers as channels and can define new channels.

We use the monitoring channel class for system-level profiling. The data transfer of the monitoring channel consists of a write access from a data sending process and a read access from a data receiving process. The monitoring channel performs as follows;

1. Wait for read access and write access.
2. Read the written data.
3. Report the current time.
4. Wait for one unit of time.
5. Notify the completion of the write access.
6. Notify the completion of the read access, sending the written data.

After the access to the monitoring channel, the data sending process and the data receiving process wait until notification from the monitoring channel.

The data transfer timing of each channel can be profiled by compiling SystemC codes with the monitoring channels and executing the binary code generated from compiling.

4.2. Construction of system-level execution dependency graph

This section denotes the method to construct the System-level Execution Dependency Graph (SL-EDG).

The SL-EDG contains vertices representing data transfers and processings, and edges representing dependencies of the system-level model. Each data transfer vertex contains the data size of the translated data.

The SL-EDG construction flow is as follows;

1. Make vertices representing each of the data processings, and make an edge from the *n*-th data processing vertex to the *n+1*-th data processing vertex.
2. Make vertices with the data size representing each of the data transfers, and make an edge from the *n*-th data transfer vertex to the *n+1*-th data transfer vertex.
3. Make an edge representing the **R/E Dependency** from the data transfer vertex representing data receiving for the *n*-th data processing to the *n*-th data processing.
4. Make an edge representing the **E/S Dependency** from the *n*-th data processing to the data transfer vertex representing the data sending of the *n*-th data processing.

It is possible to construct the AL-EDGs for various target architectures from the same SL-EDG, therefore, the SL-EDG should only be constructed once.

4.3. Construction of architecture-level execution dependency graph

Construction of the AL-EDG is adding edges representing dependencies of the architecture-level model to the SL-EDG. This phase needs the constructed SL-EDG and the architecture-level model consisting of the mapping information representing mappings between modules and processes and between busses and channels.

The AL-EDG construction flow is as follows;

1. Make an edge representing the **E/R Dependency** from the *n*-th data processing vertex to the data transfer vertex of the data receiving for the *n+b*-th data processing, where *b* is the number of the buffer on the port to which the channel of the data transfer is mapped.
2. Make an edge representing the **S/E Dependency** from the data transfer vertex of the data sending of the *n*-th data processing to the *n+b*-th data processing, where *b* is the number of the buffer on the port to which the channel of the data transfer is mapped.

4.4. Analysis of execution dependency graph

The AL-EDG analysis implements the system performance estimation of the target architecture. The inputs of the analysis program are the constructed AL-EDG and the architecture-level model information consisting of variables such as the bus rate and its output is the execution time of the system on the target architecture.

Before starting the analysis, the program calculates the execution time of each AL-EDG vertex. The execution time of a channel vertex is calculated from the data size and the bit-width and the rate of the bus that the channel is mapped to, and the execution time of the data processing vertex of the process is set to the execution time of the module to which the process is mapped. The execution time of the module is stored in the IP database.

The analysis flow of the AL-EDG is as follows;

1. Initialize time variable T.
2. Search executable vertices that have no edge from other vertices.
3. Divide executable vertices into groups for each module or each bus that they are mapped to.
4. Select one vertex with the highest priority from each group. (Priorities of the vertices change, if the module or bus need to change priorities of the data transfers or processings.)
5. Analyze the execution of the selected vertices by the following procedure.
   (a) Let time variable $T_{shortest}$ be the shortest execution time of selected vertices.
   (b) Add the value of $T_{shortest}$ to T.
   (c) Subtract the value of $T_{shortest}$ from the execution time of all selected vertices.
   (d) Remove the vertices whose execution times become zero and the edges from these removed vertices. The vertices whose execution times are zero represent the data processings or transfers that already finished, therefore, the value of T at that time is the finish time of the data processing or transfer of these vertices.
6. Continue 2. to 5. until all vertices have been removed.

The value of T at the time when all vertices are removed means the finish time of the system execution. Thus, the final value of T represents the estimated execution time of the system.

5. Experiment

This section shows experimental results to present the effectiveness of our method. We implemented 3 programs, the SL-EDG construction program, the AL-EDG construction program and the AL-EDG analysis program, and compared the amount of the time for the estimation of our method and the architecture-level simulation.

5.1. Target system

We applied our method to an audio/video encoding system. Figure 4 shows the system-level model of the audio/video encoding system. The system-level model consists of ten processes, three RAMs (RAM_A, RAM_V and RAM_AV), an Audio Interface (AI), an MP3 Encoder (MP3), a CCD, an LCD, a JPEG Encoder (JPEG), an AV Multiplier(AV_MUL) and a Storage Device (STRG), and nine channels, c1 to c9.

In this experiment, we estimated the execution time for encoding a 30 second piece of audio and video data at eight architecture-level models consisting of different modules, bus topologies, and bus data transfer rates. Figure 5 shows

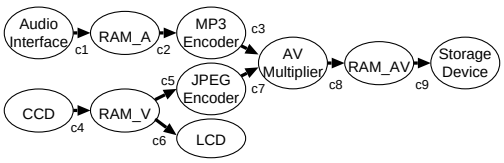

Figure 4. The SLM of the target system.

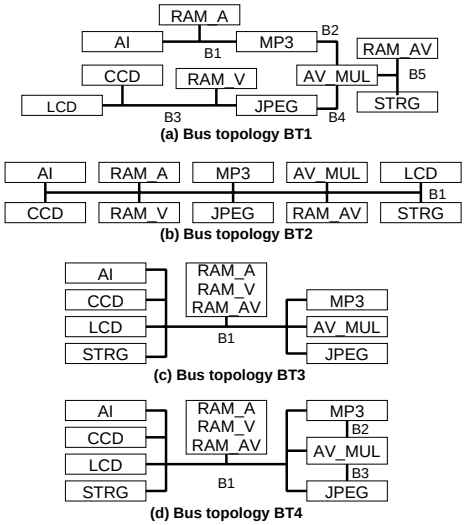

Figure 5. Bus topologies of the target system.

the four topologies. The bit-width of the buses is 24 bits. Each process except RAMs is mapped to a module one-on-one. Topology BT1 and BT2 have three RAMs to which each RAM process is mapped, and topology BT3 and BT4 have one RAM to which all RAM processes are mapped. The execution time of the MP3 Encoder, the JPEG Encoder, the RAMs, the AV Multiplier and other modules are 9021 $[\mu s]$, 6908 $[\mu s]$, 4 $[\mu s]$, 1000 $[\mu s]$ and 1 $[\mu s]$, respectively that are obtained from the IP database. Furthermore, Table 1 shows the target architecture information consisting of the bus topology and the bus data transfer rates.

Arch. name	Bus topology	Bus rate $[ns/data]$	
ARCH1	BT1	All	: 66
ARCH2	BT2	All	: 66
ARCH3	BT3	All	: 66
ARCH4	BT4	All	: 66
ARCH5	BT1	All	: 100
ARCH6	BT1	B1, B2, B4, B5	: 200
		B3	: 66
ARCH7	BT4	All	: 100
ARCH8	BT4	B1	: 66
		B2, B3	: 200

Table 1. Target architectures' information.

5.2. Comparing results of estimation time

In this experiment, we assume the data transfer time, $transTime[sec]$ by the following equation where $size[bits]$ is the data size, $bw[bits]$ is the bit-width of the bus, and the $rate[sec/data]$ is the time to send bw bits data.

$$transTime = \left\lceil \frac{size}{bw} \right\rceil * rate \quad (1)$$

We made SystemC description for system-level profiling and for architecture-level simulation that data transfer time is based on equation (1). They have same description of process behavior, but are different in fineness of data transfer. The description for only profiling has channel description, while that for architecture-level simulation has bus description that simulates data transfers and arbitration. This experiment was undertaken in a Pentium 4 (2 GHz), 512 MB memory and GNU compiler gcc-2.96 environment.

Table 2 shows the estimation time per architecture obtained by our method and those obtained by the architecture-level simulation. The estimation time per architecture means the average of the time to estimate the performance of the eight architectures shown in Table 1. Table 2 shows that our method can estimate the architecture performance about two times faster than the architecture-level simulation. In this experiment, the SL-EDG contains 17696 vertices and 23070 edges.

In our method, the compilation and simulation of SystemC descriptions for the profiling need not be executed for each architecture. Thus, we shows the estimation time for n architectures in Table 2. If many architectures' performance are evaluated, our method is about 10,000 times faster than the architecture-level simulation.

The architecture-level simulation is performed by monitoring all processes and channels even if these are idling, but our method monitors only processes and channels that are operating. Therefore, our method can estimate the system performance faster than the architecture-level simulation. Moreover, in the architecture-level simulation, designers should write the architecture-level SystemC description for each target architecture and should repeat the architecture-level simulation. In contrast, our method needs the system-level SystemC description and the one time simulation. The more target architectures there are, the bigger advantages our method has.

		1 arch.[sec]	n arch.[sec]
Architecture-level		1123.05	1123.05 * n
Our method	Sim. + SL-EDG const.	687.74	687.74 + 0.09 * n
	AL-EDG const. + analy.	0.09	
	Total	687.83	

Table 2. Estimation time of two methods.

6. Conclusions

In this paper, we proposed architecture-level estimation method based on the EDG construction and EDG analysis. Experimental results show that our method can estimate the architecture performance faster than the architecture-level simulation.

In future, we plan to propose a system-level architecture exploration method.

Acknowledgment

We would like to express our special thanks to Prof. Jun Sato of Tsuruoka National College of Technology, Dr. Shinsuke Kobayashi of University of Tokyo and Mr. Noboru Yoneoka at Integrated System Design Laboratory of Osaka University. This work is partly supported by STARC (Semiconductor Technology Academic Research Center) as research number 202.

References

[1] F. Balarin, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, Y. Watanabe, and G. Yang. Concurrent execution semantics and sequential simulation algorithms for the Metropolis meta-model. In *Proc. of the 10th International Symposium on Hardware/Software Codesign (CODES2002)*, pages 13–18, 2002.

[2] D. D. Gajski. IP-based methodology. In *Proc. of 36th Design Automation Conference (DAC1999)*, page 43, Jun. 1999.

[3] K. Lahiri, A. Raghunathan, and S. Dey. System-level performance analysis for designing on-chip communication architectures. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 20(6):768–783, Jun. 2001.

[4] E. A. Lee. Overview of the Ptolemy project. *Technical Memorandum UCB/ERL M01/11*, Mar. 2001.

[5] T. Morifuji, Y. Takeuchi, J. Sato, and M. Imai. Flexible hardware model: Implementation and effectiveness. In *Proc. of the seventh Workshop on Synthesis and System Integration of Mixed Technologies (SASIMI'97)*, pages 83–89, Dec. 1997.

[6] M. Takahashi, H. Miyajima, and M. Fukui. An efficient power and performance evaluation method with reconfigurable bus architecture model. In *Proc. of the 11th Workshop on Synthesis and System Integration of Mixed Information Technologies (SASIMI2003)*, pages 345–350, Apr. 2003.

[7] S. Yoo, G. Nicolescu, L. Gauthier, and A. A. Jerraya. Automation generation of fast timed simulation models for operating systems in soc design. In *Proc. of the Design Automation and Test in Europe 2002 (DATE 2002)*, pages 620–627, Mar. 2002.

[8] *SystemC Version 2.0 User's Guide*. http://www.systemc.org/.