Architecture Evaluation Based on the Datapath Structure and Parallel Constraint

Masayuki YAMAGUCHI[†, ††] Akihisa YAMADA[†] Toshihiro NAKAOKA[†] Takashi KAMBE[†]

[†]Precision Technology Development Center, SHARP Corporation
2613-1 Ichinomoto-cho, Tenri, Nara, 632 Japan
Phone: +81-7436-5-2531, Fax: +81-7436-5-4968
E-mail: {masa,yamada,toshi,kambe}@edag.ptdg.sharp.co.jp

[††]Dept. Information Systems Eng., Osaka University
2-1 Yamada-Oka, Suita, Osaka, 565 Japan
Phone: +81-6-879-7807, Fax: +81-6-875-5902
E-mail: yamaguti@ise.eng.osaka-u.ac.jp

Abstract— This paper presents a novel way of evaluating architecture of embedded custom DSPs which helps designers optimizing the datapath configuration and the instruction set. Given a datapath structure, it evaluates the performance in terms of an estimated number of steps to execute the target program on the datapath. A concept of "parallel constraint" is newly introduced, which enables evaluation of the impact of instruction format design on the performance without explicitly specifying the instruction format. The number of execution steps is estimated by a combination of static analysis and dynamic analysis. It enables fast and precise estimation of actual performance in the early design stage. We show some experimental results on an actual signal processor to demonstrate the accuracy of estimation and the usefulness of this method in architecture design.

I. Introduction

Design of datapath configuration and instruction set involves the most important decisions in custom DSP design. Namely, (1) the types and the numbers of functional units, (2) the connection between them, and (3) the way the functional and transfer resources are controlled, are the decisive factors that have a great impact on the performance. Although we can not see the final performance figure until we finish the detailed design, modification of the architectural decisions at a later design stage causes a fatal delay in the whole design process. Thus, evaluation of the architectural decisions in the early design stage is of high importance in the custom DSP design.

There have been a lot of researches on performance evaluation of architectures because it is a key issue not only in architecture design, but also in high-level synthesis, hardware/software codesign [1-11], etc.

PEAS [1], COSYMA [2], and ASIA [3] are examples of architecture evaluation methods in the area of high-level synthesis and hardware/software codesign. Most of the methods in this category focus their attention on functional units. They assume simple or fixed ways of data transfer and do not adequately cover variations of such methods. Since the configuration of data transfer resources has as much influence as the selection of functional units on the performance of custom embedded processors, straightforward application of the above methods does not work well in our application.

COACH [4], POLIS [5], and CHINDERELLA [6] can be categorized as processor level or instruction level evaluation methods. COACH estimates performance by executing programs at the instruction level using compiler generation. POLIS extracts parameters from existing compiler and benchmark programs, and apply them to its parametric evaluation model. CHINDERELLA evaluates the upper and lower bounds of the number of the execution steps by program analysis. However, in the custom processor design, the instruction set is designed after the datapath design is finished. Therefore, the use of these methods in the early stage of datapath design would require the designer to make many decisions about the instruction set before it is appropriate to do so.

A research by Gong et al. [11] is the closest one to our requirement. From a given application program and a datapath configuration, it estimates the number of necessary steps to execute the program on the datapath using a retargetable scheduler. However, it assumes restricted types of instruction formats and does not deal with the variation of instruction formats. Since exploitation of instruction formats is another important factor for better cost-performance of the processor, we further need a way to evaluate the impact of the instruction format design.

In this paper, we propose a new architecture evaluation method which supports both datapath design and instruction set design. Given an performance of the datapath in terms of the number of the steps to execute the program. The distinctive features of this method are as follows:

(1) Precise performance estimation taking the cost of data transfer into account.

(2) Modeling of controls and instruction sets in terms of "parallel constraint" on the datapath configuration, which enables flexible description of the impact of the instruction format and the control method.

(3) Fast and precise estimation of performance using static analysis (scheduling of the program onto the datapath) and dynamic analysis (profiling of the program with actual data) in combination.

The hardware model in our system includes data transfer resources, such as interconnections, buses, and multiplexors, as well as functional resources to execute arithmetic/logical operations. The number of control steps for

ASP-DAC '97
0-89791-851-7$5.00

transferring data is also counted. Conflicts on buses and multiplexors are also taken into account.

Unlike COACH, our system does not take an explicit specification of the instruction set as part of the hardware model. Instead of specifying the instruction set explicitly, we introduce a new concept of a "parallel constraint". It represents, in the form of a Boolean formula, the condition where a certain combination of operational and transfer resources in the datapath cannot be activated at the same time. This provides a flexible way of specifying the influence of the instruction format on the datapath. In the early design stage where we are only interested in the datapath configuration, we specify no parallel constraint. Later we gradually add constraints until we form the instruction format. This approach differentiates our system from the existing architecture evaluation systems [4,11]. Although a similar concept is used in code generation [12], there have been no attempts to introduce this idea in architectural performance evaluation.

We have developed an architecture design support system based on the proposed method. We show its effectiveness by applying it to an actual LSI design such as an audio signal processor.

The rest of this paper is organized as follows. In section 2, software and hardware models and an overview of the system are presented. Our evaluation algorithm based on the models is described in section 3. The evaluation system is outlined in Section 4. In section 5, some experimental results using an actual design are also shown to demonstrate the effectiveness of our approach. Finally, we discuss the status and conclude with future directions.

II. Software and Hardware Models

Our system estimates the number of control steps from a given application program, a run time data, and a datapath structure (Fig. 1). We formulate the underlying software and hardware models in this section.

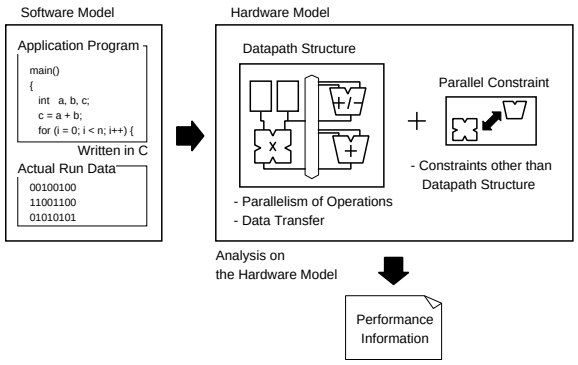

Fig. 1. Outline of the evaluation system.

A. Software Model

We assume that an input application program is written in a subset of KR type C language.

As for data types, integer, float and its array types are assumed. Other types such as string, pointer, enumerate, and complex types are not allowed. Accordingly, logical and arithmetic operators are supported. The program may consist of assign statements, conditional statements (*if-then-else*), loop statements (*for*, *while*, and *do*), procedure calls, and *return* statements.

Instead of manipulating the C code directly, we work on an abstract data structure named "BBS (Basic Block Structure)". BBS consists of a set of *basic blocks* and a *control relation*. Fig. 2 shows an example of an application program and its BBS.

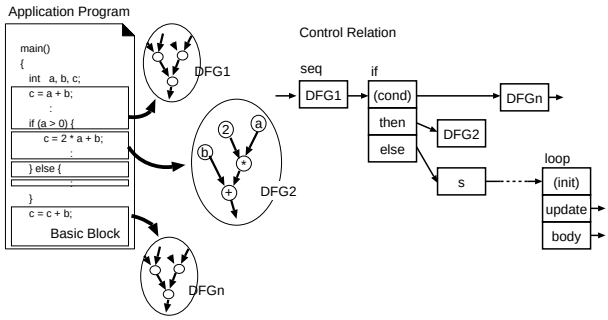

Fig. 2. An application program and BBS.

The basic block corresponds to a part of the application program which includes no branches. Each basic block is formulated as a data flow graph (henceforth abbreviated simply to a DFG). A node in the DFG represents a primitive operation executed by a hardware. The primitive operations include logical and arithmetic operations, read/write of ROM/RAM, reference to a constant. A directed edge (u, v) represents a data dependency meaning that the result of operation u is used in operation v.

The control relation between basic blocks is represented as a directed graph which corresponds to the structure of C program such as sequencing, conditional branching, looping, function calling constructs.

B. Hardware Model

In our method, the hardware is modeled by *Datapath Structure*, and *Parallel Constraint*.

B.1. Datapath Structure

A datapath structure consists of resources and interconnections. Formally, a datapath structure DP is a 4-tuple (R, P_I, P_O, I) where R is a set of resources, P_I a set of input ports, P_O a set of output ports, and I a set of interconnections. A resource $r \in R$ is a 5-tuple $(OP_r, IN_r, OUT_r, oe_r, pi_r)$. OP_r represents a set of operations which r can execute. $IN_r \subseteq P_I$ is the set of the input ports of r and $OUT_r \subseteq P_O$ the set of the output ports of r. We allow multi-cycle and pipelined operations. oe_r is the number of execution steps and pi_r is the interval of pipelining. The set of interconnections $I \subseteq P_O \times P_I$ is a relation where $(p, q) \in I$ stands for the existence of a connection from port p to port q. Fig. 3 shows an example of the datapath structure and its corresponding instance of DP.

B.2. Parallel Constraint

A parallel constraint is a constraint posed on the datapath structure, which limits the simultaneous activities of the

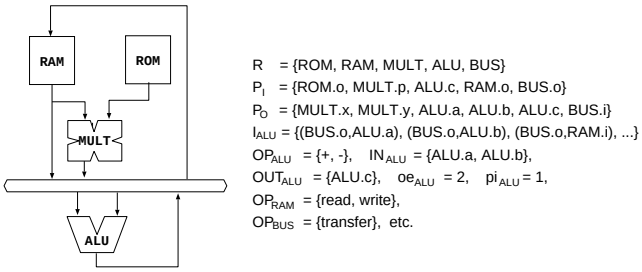

Fig. 3. A datapath and its DP representation.

resources and interconnections. The limitation is brought about by the instruction format and control structure.

Let x_r ($r \in R$) and y_i ($i \in I$) be Boolean variables where $x_r = 1$ ($y_i = 1$) *iff* r (i) is activated. Then the parallel constraint is expressed in the form of a logical expression PC consisting from x_r and y_i. It declares that activities which makes $PC = 1$ is inhibited.

For example, $x_{\mathrm{ALU}} \wedge x_{\mathrm{MULT}}$ means that ALU and MULT can not operate at the same time. Similarly, $(x_{\mathrm{ALU}} \wedge y_{(\mathrm{RAM.o,BUS.i})}) \vee (y_{(\mathrm{RAM.o,MULT.x})} \wedge y_{(\mathrm{ALU.c,BUS.i})})$ indicates that ALU operation and the data transfer from RAM.o to BUS.i as well as the data transfer from RAM.o to MULT.x and from ALU.c to BUS.i may not occur simultaneously.

The parallel constraint is useful in specifying the limitation of parallelism brought about by instruction formats. Let us see two instruction formats in Fig. 4. In VLIW type Instruction Format 1, control of ADD, MULT, MEM1, and MEM2 are assigned to distinct fields and hence all the operations may be executed in parallel. On the other hand, in shorter Instruction Format 2, ADD and MULT, and also MEM1 and MEM2, share the same instruction fields, and parallel activation of ADD and MULT, and MEM1 and MEM2 are disabled. We can represent this restriction by a parallel constraint $(x_{\mathrm{ALU}} \wedge x_{\mathrm{MULT}}) \vee (x_{\mathrm{MEM1}} \wedge x_{\mathrm{MEM2}})$.

(Instruction Format 1)

ADD control	MULT control	MEM1 Control	MEM2 Control	...

(Instruction Format 2)

0	ADD control	MEM1 Control	...
1	MULT control	MEM2 Control	...

Fig. 4. Example of instruction formats.

In the earliest design stage, we only have a configuration of the datapath where all the resources and the interconnections can operate in parallel. As the design of the instruction set and the control proceeds, however, various constraints on simultaneous operation will be posed one after another. We can express them in the form of the parallel constraint and carry out performance evaluation at any stage of the design.

III. Evaluation Method

A. Outline of Evaluation Flow

Our evaluation method consists of two major processes: *static analysis* and *dynamic analysis*.

In the static analysis, the number of the control steps is estimated for each basic block in the given application program. This is done by scheduling and mapping the DFG of the basic block onto the given datapath configuration and counting the resultant number of the control steps. In the dynamic analysis, on the other hand, the execution count of each basic block is determined by investigating the trace of the program on the given run data. Let S_i and X_i be the number of control steps and the execution count, respectively, of i-th basic block. Then the total number of steps is estimated as $\sum_i (S_i X_i)$.

B. Static Analysis

The goal of this step is to find a scheduling of the DFG of a basic block so that we can estimate the number of necessary control steps of the basic block. The quality of scheduling, in terms of the resulting number of control steps, should be as high as possible so that it gives a good estimation of what will be done by a possible optimizing compiler or by manual coding in the future. At the same time, the scheduling should be computed quickly, enabling designers to consult the evaluator every time they make a slight change on their architectures.

Our scheduling problem is very similar to ones in the area of high-level synthesis but lays much more stress on the cost of data transfers. Since the data transfer paths are given as a part of datapath specification, the number of steps consumed by data transfers is one of the most important issues in architecture evaluation. We schedule data transfers and allocate them to data transfer paths as well as we schedule and allocate operations to computational resources.

We also support such implementation techniques as multi-functional units, multi-cycle operations, pipelined functional units, chaining of operations, so as to make the estimation as accurate as possible.

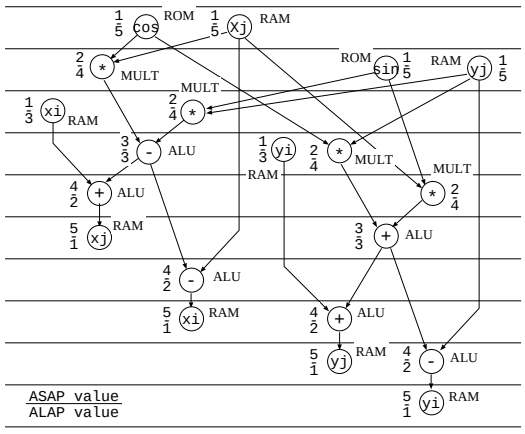

Fig. 5. The result of the first scheduling.

The scheduling problem is a type of resource constrained scheduling and we solve this by a list scheduling method [13]. In order to deal with data transfer and parallel constraint, however, we adopt two phase scheduling.

In the first phase, preliminary scheduling is done focusing only on operational resources. Each node in the DFG

is scheduled and allocated to the best possible resource in an order determined by a priority function calculated from the ASAP (As Soon As Possible) and ALAP (As Late As Possible) values of the nodes. If there are multiple resource candidates for an operation, the resource with the minimum transfer cost is selected. If the minimum candidate is already allocated to another operation, there are two choices; take the second minimum resource or delay the operation until the minimum resource become available. The decision is made based on a certain evaluation function. Fig. 5 shows a result of the first scheduling for a fraction of FFT algorithm to the data path shown in Fig. 3.

In the second phase, complete scheduling of all the operations and the data transfer is determined. *Transfer nodes* which represent the data transfer between operations are added to the DFG. The transfer nodes are scheduled and allocated to appropriate data transfer paths, while the result of the first scheduling on the operational nodes are fixed. If two operational nodes o_1 and o_2 are allocated to r_1 and r_2 in the first phase scheduling, then the node corresponding to transfer (o_1, o_2) is allocated to one of the best possible path between r_1 and r_2. The shortest path algorithm is used to find the path of the minimum transfer cost. If a conflict on buses and multiplexors are unavoidable, the data transfer and the subsequent operations are delayed. The parallel constraint is checked every time a resource is required in the scheduling. Boolean values in the constraint are updated whenever a DFG node is scheduled.

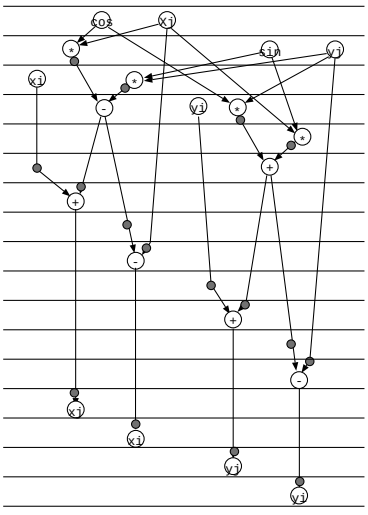

Fig. 6. The final result of the scheduling.

Fig. 6 shows the result after the second scheduling for DFG with transfer nodes (no parallel constraint is assumed in this example).

We do not pay attention to the exact location and the capacity of registers; we assume implicit registers are available at every input/output port of the resources. We make this assumption because we consider this situation corresponds to the optimized allocation and placement of registers.

IV. Overview of the Evaluation System

We have developed an architecture design support system which includes performance evaluation based on the proposal method. It is implemented in C, C++, yacc, and lex on a Sun Sparc Station 10. The evaluation flow in our system is shown in Fig. 7.

The evaluation system consists of five subprocesses: (1) hardware modeling, (2) software modeling, (3) static analysis, (4) dynamic analysis and (5) data collection.

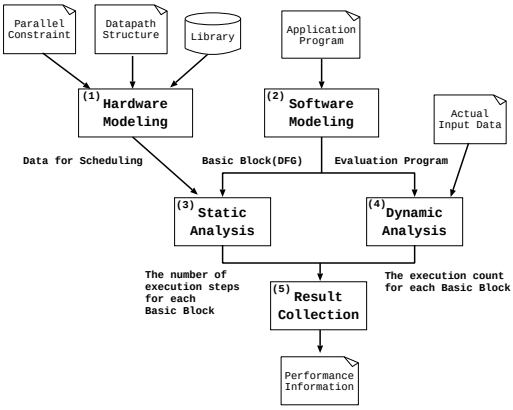

Fig. 7. Evaluation flow in our system.

In the hardware modeling process, a given datapath configuration and parallel constraint are translated into data structures DP and PC, respectively. PC is represented in the form of sum-of-products.

Both BBS and an evaluation program are generated in the software modeling process from the given application program by a source code analyzer. Every basic block is translated to a DFG. The evaluation program is a C program which is used in the dynamic analysis. Codes for counting execution frequency of each basic block and tracing the execution path are put into the original application program.

The static analysis estimates the number of the control steps for each basic block in the given application program according to the method described in the previous section. The output of the dynamic analysis is the execution count for each basic block and the trace list. The evaluation program is compiled and executed with the actual run data in this section.

The total number of execution steps is calculated by combining the results of the static and dynamic analyses in the result collection process. Statistical information such as the working ratio of each resource is computed. We have also developed a graphical trace viewer which displays the operation status of each resources, power consumption, etc; along the execution steps from beginning to end (upper right window in Fig. 8). From this information, we tell the parallelism of active resources, conflicts of data transfers, and possible bottlenecks.

Fig. 8 shows an outlook of our system, where the block diagram editor to input datapath configurations and parallel constraint, text editor to input application programs, and the viewer to show analysis results are displayed.

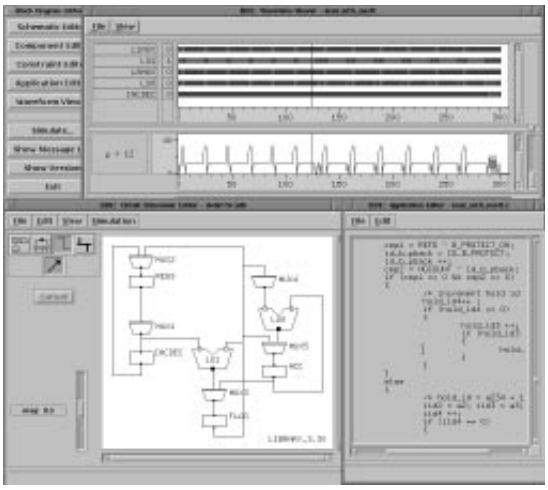

Fig. 8. The outlook of our evaluation system.

V. Experimental Results

A. Accuracy of the Estimation

In order to evaluate the accuracy of the estimation, we made an experiment on an audio signal processor which is embedded in an actual audio player. We gave to our system the same application programs as used in the audio player and the same datapath configuration and compared the resulting estimation of the instruction count with the actual instruction count.

We tried major two parts PP and FFT of the audio compression / decompression process: PP is a preprocessing part containing many branches to handle input data and FFT is a computationally intensive part containing FFT processing. A part of DFG representation of FFT is shown in Fig. 5.

The datapath structure of the audio signal processor is shown in Fig. 9 (the part in the broken line). The audio processor has adopted VLIW (Very Long Instruction Word) instruction format, and all the operators are controllable in parallel. Therefore, the parallel constraint is not set in this experiment.

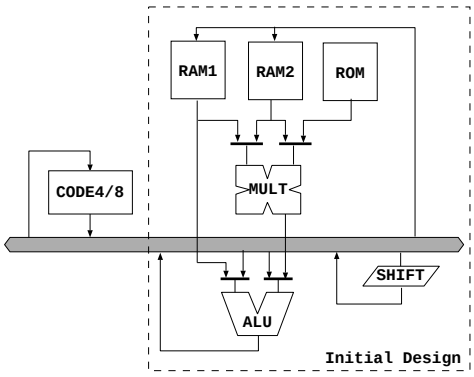

Fig. 9. Datapath structure of an audio signal processor.

Table I shows the result. #LINE and #BB show the number of lines and the number of basic blocks in the application program, respectively. EVAL is the estimated number of execution steps by our system. ACTUAL is the actual number of execution steps obtained by manual assembler coding. ERROR shows the error ratio of EVAL to ACTUAL.

TABLE I
Experimental Result of Estimation Precision

	Application Program		
	FFT	PP(1)	PP(2)
#LINE	559	636	636
#BB	66	129	129
EVAL	4,000	5,693	17,749
ACTUAL	4,750	6,057	20,653
ERROR(%)	−11.1	−6.8	−14.1

The precision error of our estimation is at most −14 % to the actual number of steps. We think these figures are fully practical as a prediction in the early architecture design stage.

B. The Effect of Parallel Constraint

In this experiment, we show how the parallel constraint can be utilized in instruction set design.

Suppose that we must re-design the instruction set of the audio signal processor in the previous experiment using a short instruction word format instead of the VLIW format. We can foresee the effect of various combinations of instruction field sharing using the parallel constraint.

TABLE II
The Effect of Parallel Constraint

Constraint	Application Program		
	FFT	PP(1)	PP(2)
NO-PC	4,000(100%)	5,693(100%)	17,749(100%)
PC1	4,936(123%)	5,693(100%)	17,749(100%)
PC2	5,184(129%)	5,843(103%)	18,671(105%)
PC3	4,131(103%)	6,343(111%)	20,407(115%)

Table II shows the numbers of estimated execution steps under no constraints (NO-PC) and various constraints of exclusiveness between the following resources:

PC1: MULT–ALU transfer and BUS transfers,
PC2: ALU operations and BUS transfers, and
PC3: RAM1 transfers and BUS transfers.

Each figure in the parenthesis shows the percentage to the NO-PC case. For example, PC1 means the MULT–ALU and BUS transfers can not be executed simultaneously because the instruction fields of them are shared.

We see PC1 and PC2 have a big influence on FFT, whereas only PC3 has a big influence on PP. Moreover, we also see the constraint PC1 does not affect the execution of PP at all. This means that the independent bus between MULT–ALU does not increase the performance of PP execution. Therefore, in case we design a DSP for only PP, we can share the control fields between BUS transfer and MULT–ALU/RAM1 transfer with a very small loss. In case of DSP for executing FFT and PP subsequently, we should better choose PC3 than PC2 for the performance.

Because the processor is actual design, we did not obtain actual numbers of execution steps of different instruction format. However, for example, we confirmed the

two facts; (a) FFT shown in Fig. 5 includes a continuous sequences of mult–add/sub–add/sub in which data may be transferred MULT–ALU–BUS–ALU or ALU operation and BUS transfer may be activated simultaneously in the datapath shown in Fig. 9, (b) the percentage of simultaneous execution of ALU and BUS at the instruction level is more than 10%. We can tell the figures of PC1 and PC2 for FFT are reasonable from these facts.

In this way, the parallel constraint can be utilized in the architecture and the instruction set design to evaluate their influences on the performance. The designer can optimize the architecture and the instruction set with the evaluation result.

C. Design Improvement by the Evaluation

Next, we show how we can use our system for design or re-design of datapath configuration. We consider a case where we design a video compress/decompress processor by making necessary modification on the audio signal processor in the previous experiments. We take variable-length code decoding (VLD) as a target application.

At first, given the target application program and the datapath structure of the audio signal processor as the initial architecture, we evaluate the performance. The first column in Table III shows its result. Suppose that the estimation does not satisfy the required performance and we must improve the architecture.

In this experiment, we tried to examine how much performance improvement is obtained by adding functional unit CODE4 or CODE8 to the original datapath, where CODE4 and CODE8 are dedicated hardware that decodes leading 4 bits and 8 bits, respectively, using a table.

The experiment took only slight modification on the original models. In the application program, we turn the part of the code corresponding to CODE4 or CODE8 into a subroutine. We added a functional unit to the hardware model and specified the relation between the unit and the subroutine in the model.

The result is summarized in Table III. The number of execution steps decreased from 634,361 steps to 419,361 steps (66.1%) with CODE4, and to 375,385 steps (59.2%) with CODE8. We may chose one of the two modifications according to the required performance.

TABLE III
PERFORMANCE IMPROVEMENT BY ADDITIONAL OPERATOR.

Architecture	Estimated steps	Improvement
Initial	634,361	100.0%
with CODE4	419,361	66.1%
with CODE8	375,385	59.2%

We also obtained the actual instruction counts (by hand assembling) of the modified hardware. They are 439,042 (70.0%) steps with CODE4 and 403,773 (64.4%) with CODE8. The errors of the figures in the table are both within 8% and we may say they are good estimations.

VI. Conclusions

In this paper, we have proposed a novel way of evaluating architecture of embedded custom DSPs. The presented method enables fast and precise estimation of actual performance at a higher abstract level. We have demonstrated the effectiveness of the method through some experiments done on a architecture design support system which we developed based on our estimation method.

One of the challenges we have to deal with is the improvement of estimation accuracy. Although the error ratios in the experiments were low, a variety of factors could be a source of unacceptable error. We are now trying to cope with pipelined control, global data flow analysis, detailed register configuration, global optimization, etc.

We are also examining the possibility of estimating the power consumption and the cost of the control part based on the same model.

Acknowledgement

The authors would like to thank Prof. Isao Shirakawa and Prof. Nagisa Ishiura of Osaka University for their constructive discussion and helpful advice on this research.

References

[1] J. Sato, Y. Honma, T. Nakata, A. Shiomi, N. Hikichi and M. Imai, "PEAS-I: A hardware/software codesign system for ASIP development," *IEICE Trans. Fundamentals, Japan*, vol. E77-A, no. 3, pp. 483–491 (March 1994).

[2] R. Ernst, J. Henkel and T. Benner, "Hardware-software cosynthesis for microcontrollers," *IEEE Design and Test of Comput.*, pp. 64–75 (Dec. 1993).

[3] I. -J. Hunag, B. Holmer and A. Despain, "ASIA: Automatic synthesis of instruction-set architectures," in *Proc. of SASIMI '93*, pp. 15–12 (Oct. 1993).

[4] H. Akaboshi and H. Yasuura, "COACH: A computer aided design tool for computer architects," *IEICE Trans. Fundamentals, Japan*, vol. E76-A, no. 10, pp. 1760–1769 (Oct. 1993).

[5] K. Suzuki and A. Sangiovanni-V., "Efficient software performance estimation methods for hardware/software codesign," in *Proc. IEEE/ACM 33rd DAC*, pp. 605–610 (1996).

[6] Y. Li and S. Malik, "Performance analysis of embedded software using implicit path enumeration," in *Proc. IEEE/ACM 32nd DAC*, pp. 456–461 (1995).

[7] R. K. Gupta and G. De Micheli, "Hardware-software cosynthesis for digital systems," *IEEE Design and Test of Comput.*, vol. 10, no. 3, pp.29-41 (Sept. 1993).

[8] M. Horowitz and K. Keutzer, "Hardware - software codesign," in *Proc. SASIMI '93*, pp. 5–14 (Oct. 1993).

[9] A. Karavade and E. A. Lee, "A hardware-software codesign methodology for DSP applications," *IEEE Design and Test of Comput.*, vol. 10, no. 3, pp. 16–28 (Sept. 1993).

[10] G. Menez, M. Augin, F. Boéri and C. Carrière, "A partitioning algorithm for system-level synthesis," in *Proc. IEEE/ACM EDAC'92*, pp. 482–487 (1992).

[11] J. Gong, D. D. Gajski and A. Nicolau, "Performance evaluation for application-specific architectures," *IEEE Trans. on VLSI*, vol. 3, no. 4, pp. 483–490 (Dec. 1995).

[12] M. Strik, J. Meerbergen, A. Timmer, J. Jess and S. Note, "Efficient code generation for in-house DSP-cores," in *Proc. IEEE EDTC. '95*, pp. 244–249 (1995).

[13] D. D. Gajski, N. Dutt, A. Wu and S. Lin, *High-Level Synthesis: Introduction to Chip and System Design*, Kluwer Academic Publishers (1992).