Trace-Driven Hybrid Simulation Methodology for Simulation Speedup : Rapid Evaluation of a Pipelined Processor

Ho Young Kim and Tag Gon Kim
Systems Modeling Simulation Lab.,
Department of Electrical Engineering and Computer Science,
Korea Advanced Institute of Science and Technology (KAIST),
373-1 Kuseong-dong, Yuseong-gu, Daejeon 305-701, Republic of Korea
hykim@smslab.kaist.ac.kr

Keywords: Hybrid simulation, trace based simulation, pipelined processor, cycle counts evaluation, simulation speed-up

ABSTRACT

This paper proposes a rapid evaluation scheme for cycle counts of a pipelined processor using hybrid simulation technique. A performance model for the simulation consists of both an analytic model and a simulation one, which takes all hazard types of pipeline architecture into consideration. The analytic model reflects structural and data hazards; the simulation model deals with control hazard. A performance simulator for pipeline architecture has been developed through which greater speedup has been made compared with other approaches in evaluation of cycle counts.

I. INTRODUCTION

Appearance of totally new applications accelerates the development of new embedded systems associated with them. Moreover, specification of such applications evolves over time, thus making programmability of the systems more and more important. Therefore, a processor component should be developed in such a way that both performance satisfying the requirement and fast time-to-market are achieved. Since exploration of an optimal processor for an application is a time intensive task, a cycle-accurate processor simulator for fast performance evaluation is crucial.

Simulation of pipelined processors involves different levels of abstraction from a gate level to a functional level. Simulation for measurement of accurate cycle counts requires a abstraction level which is lower than or equal to the cycle-accurate level. Unfortunately, development of a cycle-accurate simulator for a given microarchitecture is time-consuming and error-prone. Furthermore, simulation time for such performance evaluation is very slow because of the complex pipeline mechanism.

Processor pipelining is an implementation technique whereby multiple instructions are overlapped in their execution. For the purpose of pipelining, an instruction is divided into a sequence of stages such as fetch(F), decode(D), execute(E), memory access(M) and write back(W). If all instructions of an application program are fully pipelined, CPI (cycle per instruction) is just one. Under this ideal condition, a total cycle count of a given program is identical to the number of instructions executed within the program. In practice, however, instructions will not be perfectly pipelined; furthermore, pipelining involves some overhead occurred from pipeline hazards in three classes[9]. First, structural hazard arises from resource conflicts in which hardware cannot support simultaneous execution of all possible combinations of instructions overlapped in pipeline. Second, data hazard arises when data to be used in an instruction is obtained from a previous instruction which is overlapped in pipeline. Finally, control hazard arises from either pipelining of branches or other instructions which change an address of a next instruction.

The above hazards stall the pipeline by which execution of instructions is delayed. Due to the dynamic characteristics of the pipeline behavior an analytic performance model for an accurate analysis of all three hazards can not be obtained. However, two of the three hazards may be modeled in an analytic form. Thus, we employ a hybrid modeling approach for a better efficiency in simulation-based performance evaluation of pipeline architecture. More specifically, a hybrid model consists of both analytic and simulation models. The analytic model reflects structural and data hazards; the simulation model deals with control hazard.

This paper is organized as follows. Section II reviews related work and Section III presents a hybrid simulation model. Details of cycle count calculation within an analytic

model are described in Section IV. Experimental result is showed in Section V and conclusion is made in Section VI.

II. RELATED WORK

In the past, many attempts have been made to improve evaluation cost of processor architecture. A typical approach for accelerating the simulation of pipelined architecture is cycle-based simulation [1]. In contrast to event-driven simulation, this method fixes an execution sequence of entities that comprises the overall architecture, and invokes all the entities in that sequence. Thus, fast simulation time is achieved by removing event scheduling overhead. LISA framework[2] has improved simulation time by performing instruction decoding, operation sequencing and operation instantiation at compile time. The approach introduced static scheduling technique which reduces run-time scheduling overhead. But the framework still needs dynamic scheduling at run-time and exploration of the entire table of pipeline states. A notable technique is proposed in [3], which introduces a delay operator to solve pipeline hazards automatically. The operator is used for modeling of pipeline architecture and data hazard. However, calculation of the iterative delay operator degraded overall simulation speed. Park[4] proposed fast behavioral simulation using early pipeline evaluation technique. The approach can compute the subsequent values without consideration of data dependency between nearby instructions by separating a computing data operation from pipeline advancement. Elimination of the data forwarding mechanism made them achieve high speed-up. However, the approach needs an iterative forwarding instruction in pipeline and updating a future register, which may make simulation speed slower. Moreover, the work was limited to simple pipeline architecture of RISC processors. Binary translation is a well-known fast simulation technique[2][5] in which target processor instructions is executed by host processor instructions after translation. High simulation speed-up is achieved with difficulty in translation which preserves an accurate cycle-counts in pipeline architecture. A token-level computational model for fast pipeline simulation is also proposed[6]. The approach employs a control flow trace to eliminate overhead (evaluating the value for registers) for different pipeline architectures which has the same instruction set architecture. The approach needs only to check structural/data hazards without updating register during simulation. The proposed methodology in this paper holds this concept. However, some parts of a simulation model used in [6] would be replaced by an analytic model for further speedup without sacrifice of accuracy.

Analytic methods seek to achieve fast evaluation time at the sacrifice of evaluation accuracy. Conte[7] introduced a systematic sampling method that utilized contiguous traces of data. The result showed about 15 times faster in evaluation time than simulation by sacrificing accuracy by 13 percent. Lauterbach[8] proposed an iterative sampling-verification-resampling technique which measures the representative of partial data by instruction frequency, basic-block densities and cache statistics. With initial clusters selected at random intervals, iterative checks are performed if they represent the original benchmark. If not, new traces are inserted. This method could guarantee accuracy, but the sample is inclined to grow to the full trace. A few more works are reported in this context. Nevertheless, a question on how closely a sample trace represents the original benchmark remains as a problem.

We proposed a hybrid simulation methodology which combines analysis with simulation. The overall framework of the proposed methodology is represented in Figure 1.

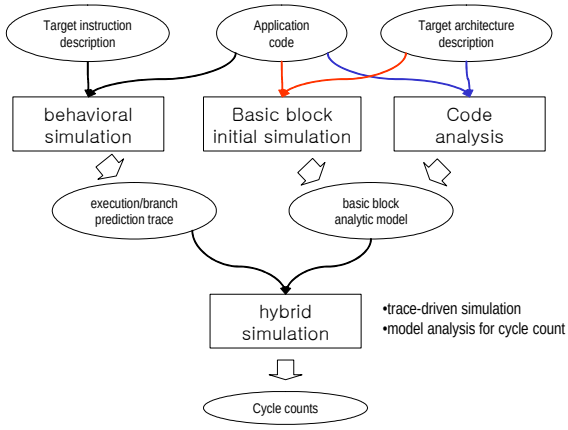

Figure 1. Overall evaluation framework

III. PROCESSOR SIMULATION MODEL

Generally, an application program is loop-intensive. As rule of thumb, a program spends 90% of its time in 10% of its code. Especially, multimedia applications perform loop-iterative data processing with massive input data. Traditional simulators are executing the same sequence of instructions in an iterative loop, which is very inefficient in simulation time. To improve it we introduce a simulation model based on a basic block. The basic block is a sequence of instruction statements without any branch. Compared to the instruction based simulation, basic block simulation can reduce iterative redundant operations, thereby achieving speedup in evaluation.

For the basic block based simulation, we separate register calculation from processor simulation proposed in [6]. The register calculation does not affect on pipeline scheduling except for determining control flow, trace of which can be easily acquired by behavioral simulation or host processor execution. A simulation model in our approach combines a processor architecture and an application program. This model needs parameters with which we can get latencies of functional units, inspect pipeline hazards and then calculate total cycle counts for the application. Thanks to a property of the basic block, it is guaranteed that there is no control

hazard in a basic block. This property allows us to divide processor simulation model into two parts. One is a simulation model which takes control hazard between basic blocks into consideration, and the other is an analytic model in which we can inspect structural and data hazards within a basic block. Figure 2 shows the concept of hybrid simulation proposed in our framework. During simulation, each analytic model is evaluated whenever a control flow visits it on an execution trace. The *final pipeline states(FPS)* is introduced to represents the pipeline state in the cycles when a current basic block overlaps a next basic block. Thus, a pipeline scheduling of a next basic block depends on FPS. An example of FPS is shown in Fig. 3 in a shaded area where a stalled stage is represented by a hyphen.

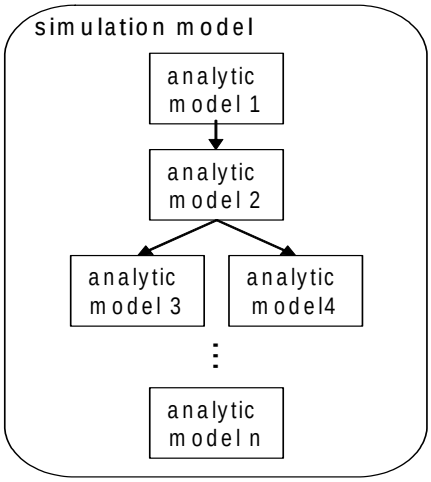

Figure 2. Concept of analytic model and simulation model

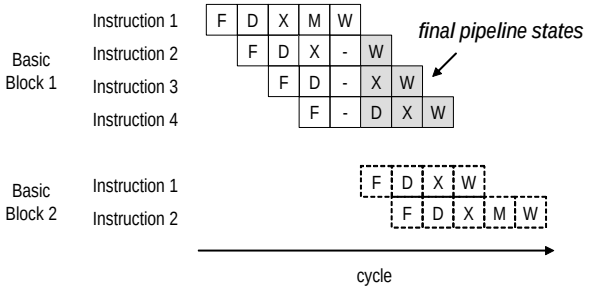

Figure 3. Final pipeline states

Analytic model – basic block and architectural model

The analytic model is an abstraction model for calculation of exact cycle counts, which is derived from a basic block and pipeline architecture.

initial pipeline states – pipeline stages occupied by instructions of the current basic block at every cycle when the instructions are simulated in the empty pipeline. Here, *initial* means the simulation starts in the empty pipeline.

initial FPS – as mentioned before, remaining instructions in the pipeline when first instruction in the next basic block comes into the pipeline. They affect on cycle counts and final pipeline states in the next basic block. Initial FPS is acquired from the initial pipeline states.

initial cycle count – execution cycle counts for the basic block during basic block initial simulation

Independent simulation of each basic block gives above information by which an analytic model for each basic block is developed.

In case of a processor with cache memory, modeling is a bit complex. Using an extra cache simulator, we should divide basic blocks in more detail. When a cache hit/miss ratio of some basic block is varying during simulation, the basic block model should be separated into different models (Figure 4).

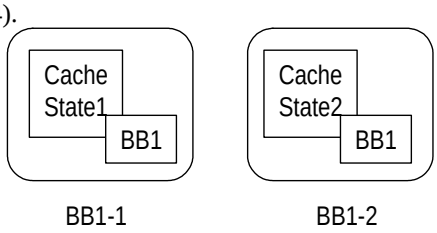

Figure 4 Basic block models classified by cache memory

Simulation model – control flow model

A simulation model is obtained from a control flow graph (CFG) of an application program. A CFG is an abstract data structure in which each node represents a basic block and a directed edge represents a jump of the control flow. Figure 5 shows a typical example of CFG. Dynamics of the model is identified by a trace of behavioral level simulation of instructions, which consists of sequences of basic blocks and branch prediction information.

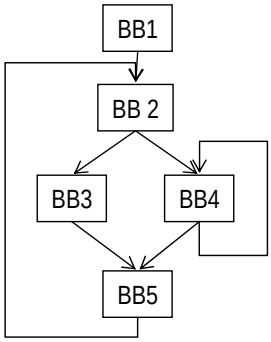

Figure 5. An example of CFG

IV. DETAILS OF THE CYCLE COUNT CALCULATION

Our main concern is fast evaluation of performance in cycle counts for a pipelined processor model. To this end, we should examine relations between basic blocks. Our entire approach hinges on an assertion:

Assertion 1
Pipeline scheduling is determined only by a current pipeline state at a cycle of fetching an instruction, assuming that cache hit/miss is same. That is, pipeline scheduling is always the same if pipeline states are identical.

During simulation, current pipeline states at the beginning of a basic block may be varying, which affect the FPS of the current basic block and execution cycle counts over the basic block. Thus we need a method for calculating cycle counts and obtaining FPS from a previous FPS. If there is no structural/data hazard brought about by the previous FPS, a cycle count and FPS are an initial cycle count and an initial FPS, respectively. If structural/data hazards occur, reschedule of the stages of instructions is needed to reflect hazards. The instructions should be rescheduled in the pipeline until new pipeline states by rescheduling are identical to the initial pipeline states at certain cycle.

From the above assertion, three cases are to be considered to obtain FPS when hazards between basic blocks occur.

1. If rescheduling occurs before final pipeline states start (see Figure 6) ,
   - FPS does not change.
   - Cycle count increases or do not change according to rescheduling
2. If structural/data hazards from the previous FPS occur at the cycle when the first instruction of the next basic block comes into the pipeline (see Figure 7),
   - FPS is newly obtained from rescheduling
   - Cycle count increases by the number of stall
3. If structural/data hazards from the previous FPS occur at the cycle after the next basic block start (see Figure 8),
   - FPS is newly obtained from rescheduling
   - Cycle count does not change

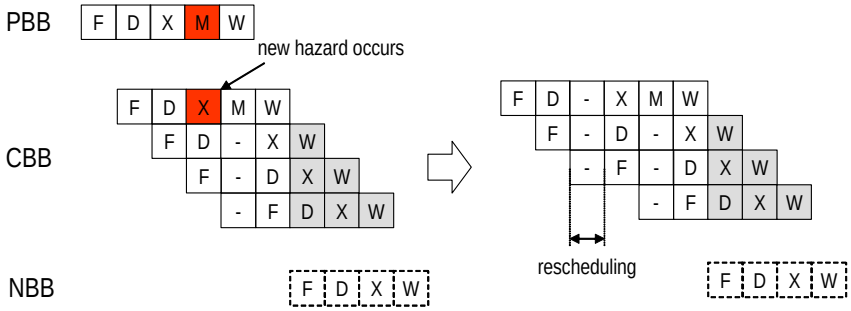

Figure 6. Basic Block Concatenation : Case I

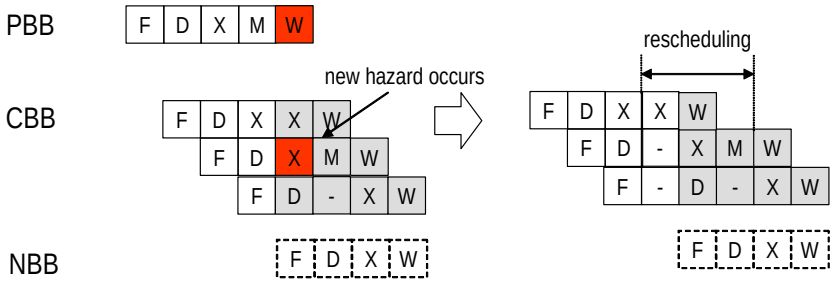

Figure 7. Basic Block Concatenation : Case II

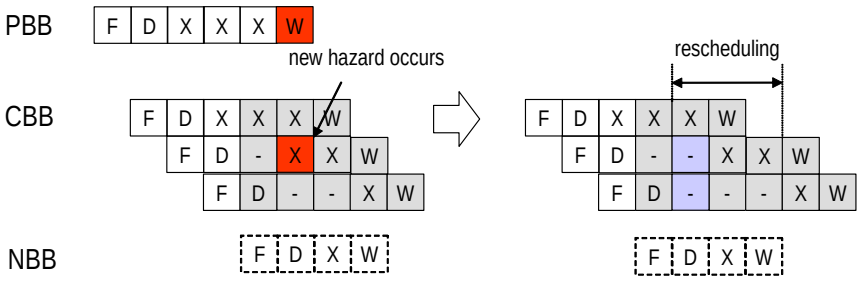

Figure 8. Basic Block Concatenation : Case III

When a control hazard occurs, a cycle count is affected by a branch prediction result. In case of branch prediction hit, no cycle count is added; otherwise a cycle count for the miss penalty should be added.

V. EXPERIMENTAL RESULT

Simulation experimentation with ARM9TDMI (no branch prediction) has been performed to show efficiency of hybrid simulation. ARM9TDMI is a high performance Thumb compatible processor, which provide a performance upgrade path from ARM7TDMI. Three application programs, namely ADPCM, IDCT and g.721, are selected from MediaBench for performance simulation. The experiment is done using AMD AthlonXP 2100+ running on Windows XP. Modeling of the processor is done by use of the XR2 architecture description language developed in our previous research[10].

Table 1 shows comparison of simulation efficiency for the proposed approach against three other approaches: instruction set simulation, token-level simulation and cycle-accurate simulation. In the table simulation results are represented in terms of MIPS (million instructions per second) and MCPS (million cycles per second). As a result, the proposed hybrid simulator has 100 times faster than a state of the art simulator.

Table 1. Comparison of simulation performance

	ISS	Token-level Simulation		Cycle-accurate Simulation		Hybrid Simulation	
	MIPS	MIPS	MCPS	MIPS	MCPS	MIPS	MCPS
adpcm	23.45	2.117	4.947	1.210	2.803	204.2	473
idct	29.045	3.192	4.753	1.924	2.818	362.5	531
g.721	21.761	2.897	4.559	1.847	2.901	270.7	426

Simulation efficiency of our hybrid simulator is greatly dependent of a processor architecture and application programs which causes different number of pipeline hazards. Moreover, no branch prediction scheme brings about stalls by control hazards, which may decrease the overlapping instructions between two basic blocks.

VI. CONCLUSION

This paper proposed a hybrid modeling approach which supports an accurate, fast simulation for evaluation of cycle counts of high-performance applications, especially loop-intensive ones. The simulation method can be applied to reduce cost of design space exploration for pipelined processors. Within the approach the analytic model eliminates iterative pipeline advancement and checking structural/data hazards, and the simulation model undertakes tracing control flow of application programs. Experimental result showed that simulation speed of the proposed approach is about 100 times faster than a state of the art simulator reported in literature. This technique can be applied not only to scalar processor but also to static scheduling ILP (Instruction Level Parallelism) processor such as a VLIW (Very Long Instruction Word) architecture.

REFERENCES

[1] Z.Barzilai, J.L. Carter, B.K.Rosen and J.D.Rutledge, "HSS : A High-Speed Simulator", *IEEE Trans. CAD/ICAS*, vol. 6, no. 4, pp.601-617, July, 1987

[2] A. Hoffmann, A.Nohl, G.Braun, O.Schliebusch, T.Kogel and H.Meyr, "A Novel Methodology for the Design of Application Specific Instruction Set Processors using a Machine Description Language", *IEEE Trans. Computer-aided Design of Integrated Circuits and Systems*, vol. 20, no. 11, pp. 1338-1354, November 2001.

[3] Felix Sheng-Ho, Chang and Alan J. Hu, "Fast Specification of Cycle-Accurate Processor Models," *in proceedings of International Conference on Computer Design*, pp. 488-492, September, 2001

[4] I. Park, S. Kang and Y Yi, "Fast cycle-accurate Behavioral Simulation for Pipelined Processors Using Early Pipeline Evaluation," *International Conference on Computer Aided Design*, pp. 138-141, Nov, 2003

[5] Jianwen Zhu et al. “An Ultra-Fast Instruction Set Simulator,” *Transaction on VLSI Systems*, vol. 10, no. 3 pp.363-373, June 2002

[6] J. K. Kim and T. G. Kim, “Trace-driven Rapid Pipeline Architecture Evaluation Scheme for ASIP Design”, *in Proc. of Asia South-Pacific Design Automation Conference*, Kitakyushu, Japan, pp.129-134, January, 2003

[7] T. M. Conte, “Systematic Computer Architecture Prototyping”, *Ph.D thesis*, Dept. of Electrical and Computer Engineering, University of Illinois, Urbana, Illinois, 1992

[8] G. Lauterbach, “Accelerating Architectural Simulation by Parallel Execution of trace samples”, *in Proc. of 27th Hawaii International Conference on System Science*, vol. 1, pp.205-210, January, 1994

[9] J. L. Hennessey and D. A. Patterson, *Computer Architecture: a quantitative approach*, Morgan Kaufmann Publisher, 1990

[10] J. K. Kim, H. Y. Kim and T. G. Kim, “Top-down Retargetable Framework with Token-level Design for Accelerating Simulation Time of Processor Architecture”, *IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences*, vol. E86-A, no. 12, pp.3089-3098, December 2003