Performance Simulation Modeling for Fast Evaluation of Pipelined Scalar Processor by Evaluation Reuse

Ho Young Kim and Tag Gon Kim
Systems Modeling Simulation Laboratory, Dept of EECS, KAIST, Korea
hykim@smslab.kaist.ac.kr, tkim@ee.kaist.ac.kr

ABSTRACT

This paper proposes a rapid and accurate evaluation scheme for cycle counts of a pipelined processor using evaluation reuse technique. Since exploration of an optimal processor is a time-consuming task due to large design space, fast evaluation methodology for an architecture is crucial. We introduce the performance simulation model which can evaluate the performance without considering the functional correctness. This model has an FSM-like form and can afford to take all hazard types of pipelined architectures into consideration. The proposed approach is based on the property that an application program, especially multimedia application, has many iterative loops in general. This property invokes many iterative operations in the simulation. Evaluation reuse scheme can alleviate redundantly iterative operations of conventional simulators in the loop. A performance simulator for the pipeline architecture has been developed through which greater speedup has been made compared with other approaches in the evaluation of cycle counts.

Categories and Subject Descriptors

I.6.0 [**Simulation and Modeling**]: General; C.0 [**General**]: Modeling of computer architecture

General Terms

Design, Language, Performance

Keywords

Retargetable simulation, compiled simulation, evaluation reuse, instruction set architecture, trace-driven simulation.

1. INTRODUCTION

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 an abstraction level which is lower than or equal to the cycle-accurate level. Unfortunately, the speed of cycle-accurate instruction set simulator is very slow because of the complex pipeline mechanism. Employing a cycle-accurate simulator is not proper for design space exploration.

There are two purposes of the processor simulation. The one is to evaluate the performance of the processor, and the other is to verify the functional correctness of the software on the processor. Therefore, we can divide the behavior of the simulator according to the purpose; functional simulation and performance simulation. Since scheduling an instruction in the pipeline depends not on data value but on the register/resource usage of it, it is possible to separate the timing simulation from the functional simulation. Table 1 shows each simulation model and the property of it compared to analytic model. We evaluate the accurate performance rapidly by use of the performance model, while we verify the correctness with functional model.

This paper proposes a new simulation technique to reduce the overhead of performance simulation. The factors affecting the cycle count are pipeline hazards[8] which occur as the instructions are scheduled. The main point is to simulate only these factors and reuse the simulation result(state transition) as possible, not to mimic the entire behaviors of the hardware.

Table 1. Property of the Simulation Models

		Evaluation Performance	Functional correctness
Analytic Model		fast, but inaccurate	not verified
Simulation Model	Traditional model	slow, accurate	verified
	Functional model	Impossible	verified
	Performance model	relatively fast, accurate	not verified

This paper is organized as follows. Section 2 reviews related work and in section 3 we propose a FSM-like performance simulation model for simulation reuse. Experimental result is showed in section 4 and conclusion is made in section 5.

2. RELATED WORK

Instruction set architecture simulation(ISS) is the highest abstract level of simulation to acquire accurate cycle count for an application. Unlike lower level simulation, the overhead of the simulation is concentrated on instruction decoding, functional operation and instruction scheduling. In the last decade, many attempts have been made to reduce above overheads.

Analytic methods seek to achieve fast evaluation time at the sacrifice of evaluation accuracy[2]. In their works, they achieved great gain of evaluation speed by sacrificing accuracy by about 10 percents. In [3], ISS with caching result was used to speed up estimation. However, above methods are not feasible in case of some complex architectures and applications. Especially, these approaches are not proper for a hard real-time application since they cannot determine the exact error bound. Compiled simulation technique has improved simulation speed which is performing instruction decoding at compile time. To improve the simulation speed further, static scheduling based techniques move the instruction scheduling into the compilation phase[1]. Even though getting great speed-up, the simulator still needs inevitable dynamic scheduling (indirect branch) and the simulator must consider every possible case that can be encountered in dynamic scheduling since the simulator attempts to resolve pipeline hazards statically at the compile time. Recently, cached compiled simulation has been developed[4]. It is useful for dynamic switching of instruction set modes or applications with run-time dynamic program code. Early pipeline evaluation technique was proposed for fast behavioral simulation[5]. The approach can compute the subsequent values without consideration of data dependency between nearby instructions. Binary translation is a well-known fast simulation technique[6] in which target processor instructions are executed by host processor instructions after translation. High simulation speedup is achieved with difficulty in translation which preserves an accurate cycle-counts in pipeline architecture. A token-level computational model for fast pipeline simulation was also proposed[7]. They proposed new abstraction level, named 'token-level', in which fast calculation for cycle count is achieved.

This paper proposes a novel simulation technique to reduce the overhead of performance simulation. Fast performance simulation can be achieved in a way that performance for each instruction in the trace is evaluated without functional verification of the instruction itself. For this, new abstract performance simulation model is introduced. Simulation reuse supports simulation speedup by elimination of redundant simulation

DAC 2005, June 13-17, 2005, Anaheim, California, USA.

3. PROPOSED SIMULATION MODEL OF A PIPELINED PROCESSOR

3.1 Pipelined processor simulation

Instruction set architecture simulator carries out roughly following operations; instruction fetch, instruction decoding and instruction execution.

1. **instruction fetch** : get an instruction from the instruction memory as the value of PC
2. **instruction decoding** : decode the fetched instruction to know the behavior, operands and addressing modes.
3. **instruction execution**
   - *functional execution* : execute the functional behavior of the decoded instruction such as arithmetic/logical operation, load/store, move, branch, etc.
   - *instruction scheduling* : check the hazard of pipeline and manage the advance of the instruction in the pipeline.

Most of the simulation time is consumed by instruction decoding and instruction execution since they have many operations of comparison for the decoding and scheduling. Therefore, the performance of a simulator heavily depends on how the instruction decoding and the instruction execution are achieved.

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. Whenever the instruction at the same memory address is executed, the simulator carries out the same operation of instruction fetch and instruction decoding. In compiled simulation, the instruction decoding is only performed once for each instruction(see figure 2), whereas interpretive simulators decode the same instruction multiple times if it is part of a loop(see figure 1). In token-level simulation, functional simulation is performed first, which generates a trace of executed instructions. For the architectures with same instruction functionality, they can evaluate the performance of the architecture only by scheduling the instructions according to the trace without functional execution of them(see figure 3).

This paper holds the concept of the compiled simulation and token-level simulation. In addition to two concepts, our work has emphasis on reducing the simulation overhead of instruction scheduling. We introduce a basic block based performance simulation model(figure 4). Compared to the conventional simulation, the basic block based simulation can reduce iterative redundant operations such as managing pipeline, thereby achieving speedup in evaluation.

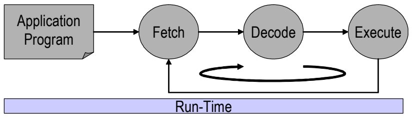


Figure 1. Interpretive simulation flow

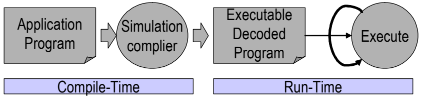


Figure 2. Compiled simulation flow

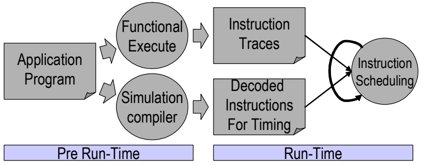


Figure 3. Token-level simulation flow

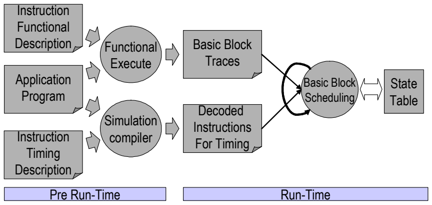


Figure 4. Proposed framework of fast evaluation

3.2 Performance Simulation Model

Though a same sequence of instructions is executed iteratively in the same loop, scheduling of the instructions still depends on the scheduling of the preceding instructions. Therefore we should define the architectural state of the processor at the highest abstract level where we can schedule cycle-accurately an instruction into the pipeline. For this, we modeled the processor behavior the finite state machine form. We considered the followings among others in our model.

- single issued scalar architecture
- inter-basic block hazard as well as intra-basic block hazard
- the instruction which has various execution stages with different durations
- multi-level instruction/data cache memory (the model is extended in section 3.3.2)

To schedule instructions in the basic block, we need only to consider the structural hazards and the data hazards since control hazard cannot happen. These two hazards arise from resource conflict and data dependency. Therefore, the state of the FSM model should include the use of registers and functional units. We introduce a scoreboard to show the use of registers and functional units at each cycle. However, scheduling an instruction is affected not by the scoreboard at a cycle, but by a set of scoreboards for a few cycles while the instruction remains in the pipeline. Therefore the state of the model is determined by a set of scoreboards which affect scheduling of a current instruction. Here is the modeling formalism

M = <R, SB, S, I, C, λ, δ>, where

- *R : union of a set of registers and a set of resources*
- *SB :* 2^R*, a set of scoreboard about the uses of registers and resources*
- *S :* 2^{SB}*, a set of scoreboards for several cycles*
- *I : a set of instructions with usage of registers and resources*
- *C :* $C \subset N$*, a set of number of cycle count (N: natural number)*
- $\lambda : S \rightarrow C$*, output function*
- $\delta : S \times I \rightarrow S$*, scheduling function*

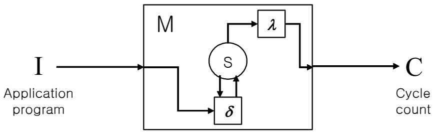
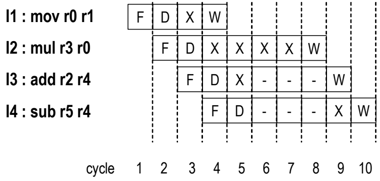


Figure 5. Semantics for performance modeling

As opposed to a real pipelined processor, the proposed model computes all pipelined hazards of an instruction and schedules the instruction in the pipeline at a time, not at cycle by cycle. Scheduling function δ denotes state transition, while λ means output function generating cycle count. In fact, state transition means scheduling of an instruction, which is resolved as inspecting and stalling the instruction if any hazard occurs. Cycle count means the cycle until the next instruction will be fetched. The example about the state transition is showed in figure 6. The processor under consideration is assumed to have a 4-stage pipeline consisting of {IF, ID, EX, WB} with data forwarding from EX to ID. We assume that *mov*, *sub* and *add* instruction take 1 cycle in ALU, while *mul* instruction takes 4 cycles in multiplier unit. 4 instructions (I1~I4) are scheduled as figure 6 shows.



	1	2	3	4	5	6	7	8	9	10
I1 : mov r0 r1	F	D	X	W						
I2 : mul r3 r0		F	D	X	X	X	X	W		
I3 : add r2 r4			F	D	X	-	-	-	W	
I4 : sub r5 r4				F	D	-	-	-	X	W

Figure 6. Scheduling instructions in the pipeline

In figure 6, hyphen(-) means the stall of the pipeline and other alphabets represent IF, ID, EX and WB, respectively. Figure 7 illustrates the state(scoreboards) transitions for each instruction. The next state is determined by timing behavior of each instruction. In figure 7, each line of ScoreBoards represents a scoreboard for each cycle. ID, ALU, MUL and WB indicate the used resources while r0~r5 represent registers. Above modeling formalism, total state space has exponential complexity, but moveable state space at simulation run-time is a tiny part of it due to reiteration of inputs.

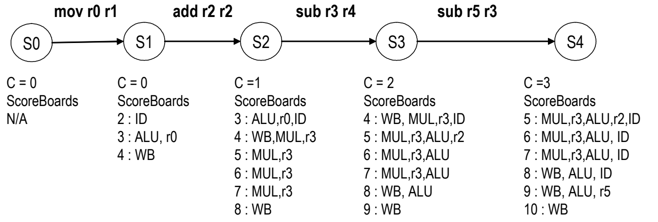


Figure 7. State transitions of figure 6

3.3 Cycle Count Calculation with Simulation Reuse

To get an accurate cycle counts, we assume that the latencies of function units are independent of the input values. We will discuss about the instruction which has data-varying latency in section 3.4.

3.3.1 Processor without cache memory

The FSM is deterministic, that is, each combination of state and input unambiguously defines the next state to move into. Unlike general FSM, our processor model has iterative a sequence of inputs due to loop-intensive property of applications. Consequently, it has high probability of same transition, so it is possible to reuse the state transition for achieving efficient simulation. Whenever the first instruction of a basic block is going to be executed, the simulator looks up the state table entry of the basic block that corresponds to the current pipeline state. The previously stored data, such as final state and executed cycles is used to accelerate the simulation. Figure 7 illustrates the workflow of the reuse based simulation. We assume the simulator is going to execute the first instruction of the basic block 4 and current executed cycles are 384. First, the simulator looks up the initial state in the state table of the basic block 4(1) and receives that this state has not been executed before. Therefore the instruction scheduler is invoked(2) which schedules all instructions in the basic block and stores the data(initial state, final state and cycle) in the state table(3). Finally the instruction scheduler updates the current state and executed cycle of the simulator(4), which is used to check structural/data hazards between the basic blocks. If there exists any initial state same with current state, step (2) and step (3) will be omitted.

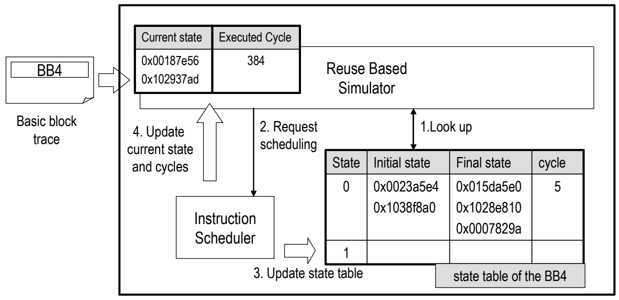


Figure 8. Simulation reuse flow without cache memory

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 and the final state should be delayed by the amount of miss penalty.

3.3.2 Processor with Cache Memory

In case of a processor with cache memory, the simulation reuse is more complex. Since scheduling an instruction is affected by cache status, i.e. cache miss requires a multi-cycle memory operation, we should consider the cache state in addition to the state of registers and resources. However, the state of the cache memory is so complex in some architectures that the state space of processors can enlarge terribly. We solved this problem with the use of an extra cache simulator such as DINERO-IV[10]. The cache simulator analyzes the overall cache state to determine whether cache hits or not. The cache hit/miss trace generated by the cache simulator, allows us to simulate the processor without considering the cache status. When load/store instructions are fetched, the latency of the memory operation is determined as the hit/miss trace. We add only cache hit/miss trace to existing state, so we can reduce the cache state space greatly. Here is extended modeling formalism.

$M = <SB, S, RG, RS, I, CT, C, \lambda, \delta>$, where

- *R : union of a set of registers and a set of resources*
- *$SB : 2^R$, a set of scoreboard about the uses of registers and resources*
- *$S : 2^{SB}$, a set of scoreboards for several cycles*
- *I : a set of instructions with usage of registers and resources*
- *$C : C \subset N$, a set of number of cycle count (N:natural number)*
- *$\lambda : S \rightarrow C$, output function*
- *$\delta : S \times I \times \{Hit, Miss\} \rightarrow S$, scheduling function*

In case of memory instruction, cycle calculation and instruction scheduling depend on cache hit/miss, while they do not care cache hit/miss in case of the other instructions.

3.4 Limitation

Though our work is concentrated on a scalar pipelined processor, we can easily extend to statically scheduled processors such as VLIW(very long instruction word) processor, whereas in case of dynamically scheduled processor, i.e. superscalar, we should consider various factors which affect the instruction scheduling, which enlarge the state space of our framework.

In some architecture, execution time of some instructions such as multiply, divide or square root, depends on data, so it is possible that the same basic block in a loop will take different number of cycles to execute. In this case, we should consider the input value of the instructions. To prevent the state space from enlarging greatly, we only add the resolved latency of the instructions to the state, not the value of data. We can get the input data of the instructions since we employed the functional simulator earlier. With the input data trace affecting the cycle count, our simulator can schedule instructions accurately and reuse the simulation result.

4. EXPERIMENT

Simulation experiment with ARM9TDMI has been performed to show the efficiency of the proposed evaluation method. ARM9TDMI is a high performance Thumb compatible processor, which provides 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 using the XR^2 architecture description language developed in our previous research[9].

Table 2 shows comparison of simulation efficiency for the proposed approach against three other approaches: instruction set behavioral simulation(ISS), token-level simulation[7] and cycle-accurate simulation(using compiled simulation technique[1]). In the table simulation results are represented in terms of MIPS (million instructions per second) and MCPS (million cycles per second). ISS is represented only by MIPS since it includes not cycle information but behavior information of the instructions, As a result, the proposed evaluation method is 4~21 times faster than a state of the art simulator.

Table 2. Comparison of Simulation Performance

		ISS	Token-level Simulation		Cycle-accurate Simulation		Proposed Evaluation	
		MIPS	MIPS	MCPS	MIPS	MCPS	MIPS	MCPS
Benchmark application	ADPCM	23.45	2.117	4.947	1.210	2.803	45.197	104.7
	IDCT	29.045	3.192	4.753	1.924	2.818	38.098	55.8
	G.721	21.761	2.897	4.559	1.847	2.901	11.715	18.4

As we know from the above tables, evaluation efficiency of proposed reuse-based simulation is highly dependent of the characteristics of application programs. The more basic blocks with distinct initial state(DBB) iterate, the more speedup will be gained. Figure 10 presents the number of instructions on each DBB and the number of iterations in each DBB. In summary, table 3 shows total reuse ratio of instruction scheduling for each application. In case of G721, the number of iterations is relatively small that we cannot get much gain for speedup, while in case of ADPCM and IDCT, the high frequency of the basic blocks enables us to evaluate the cycle count rapidly.

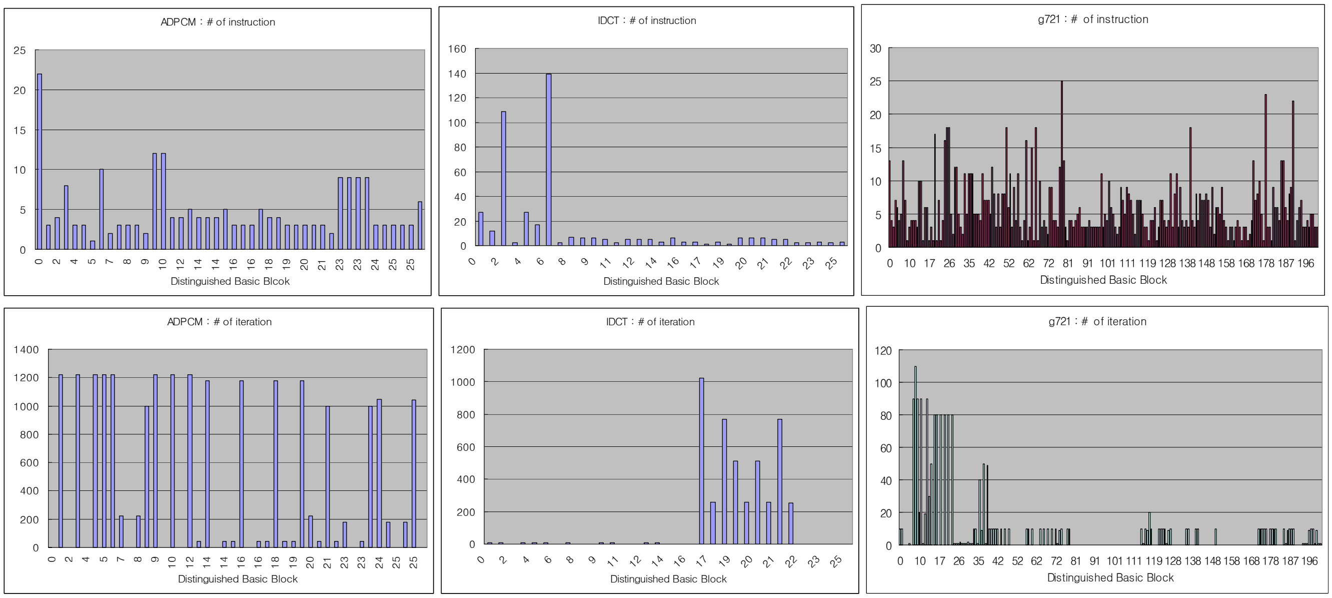


Figure 10. Characteristics of each application

Table 3. Reuse Ratio of applications

Application	# of executed instruction	# of instruction scheduling	Reuse ratio(%)
ADPCM	96667	209	99.784
IDCT	18085	188	98.961
G721	12276	724	94.102

Figure 11 shows the simulation results including cache. The cache memory under consideration is assumed to have 1-level data cache, size of 8192 bytes, 1 associative, LRU replacement and block size of 16. The experiment shows that evaluation performance does not affected greatly by cache memory. It is due to execution locality of the program and high cache hit ratio.

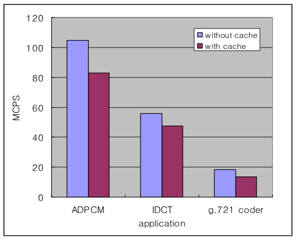


Figure 11. Simulation performance with cache

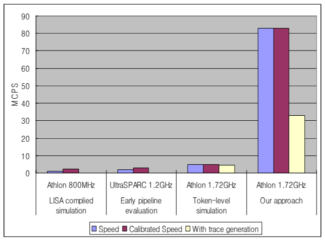


Figure 12. Comparison of other approaches

We compare the speed for ADPCM of our approach with previous works in figure 12. Since they made an experiment on the host machine which is different from ours, we calibrated their result as to the clock frequency of the CPU. The proposed approach is quite fast compared to other approaches from the figure. Though the result of our approach does not include the overhead of obtaining the basic block traces, it may not matter much since behavioral simulation is very fast compared to cycle accurate simulation.

5. CONCLUSION

This paper proposed a reuse-based evaluation approach which supports an accurate, fast simulation for evaluation of high-performance applications, especially loop-intensive ones. The evaluation method can be applied to reduce cost of design space exploration for pipelined processors. Within the approach, reusing result eliminates iterative scheduling an instruction and checking structural/data hazards. Experimental result showed that simulation speed of the proposed approach is about 4~21 times faster than a state of the art simulator reported in literature. This technique can be applied not only to scalar processors but also to static-scheduled ILP (Instruction Level Parallelism) processors such as a VLIW processor. Future work will concentrate on the system-level simulation including co-simulation as well as fast high-level power estimation.

6. REFERENCE

[1] 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.

[2] 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

[3] J. Liu, M.Lajolo and A. Sangiovanni-Vincentelli, "Software timing analysis using HW/SW cosimulation and instruction set simulator," in *Porc. Int. Workshop. Hardware-Software Codesign*, Mar. 1998, pp.65-70

[4] A. Nohl et al. "A Universal Technique for Fast and Flexible Instruction-Set Architecture Simulation", *Design Automation Conference*, pp. 22-27, June, 2002

[5] 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

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

[7] 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

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

[9] 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

[10] Mark D. Hill, "DINERO IV Trace-Driven Uniprocessor Cache Simulator", www.cs.wisc.edu/~markhill/DineroIV