Simplifying High-Frequency Microprocessor Design via Timing Constraint Speculation

Asami Tanino[1] Toshinori Sato[1,2]
[1] Department of Artificial Intelligence
[2] Center for Microelectronic Systems
Kyushu Institute of Technology
680-4 Kawazu, Iizuka, 820-8502 Japan
Phone: +81-948-29-7624, FAX:+81-948-29-7601
{asami,tsato}@mickey.ai.kyutech.ac.jp

Abstract

This paper proposes a technique to simplifying timing design of high-frequency microprocessors by speculating timing constraints. We call the technique constructive timing-violation (CTV). *Increasing clock frequency over that determined by the critical paths causes timing violations. However, if any tolerant mechanisms are provided for the timing violations, it is not necessary to keep the constraints. Rather, the violations would be constructive for high clock frequency. From these observations, we propose the CTV, which is supported by the tolerant mechanism based on contemporary speculative execution mechanisms. We evaluate the CTV and present its considerably promising potential.*
Keywords: timing constraints, speculative execution, fault tolerance

1 Introduction

Processor performance is determined by the execution time T for a program, which is given by

$$T = N * CPI * MC \quad (1)$$

where N is the number of instructions for the program, CPI is the average clock cycles per instruction, and MC is the machine clock cycle time. Based on Eq.(1), it is clear that processor performance can be improved by reducing either CPI or MC. Modern microprocessors exploits both techniques to reduce CPI and MC. The former is superscalar design, and the latter is high clock frequency. However, only latter one attracts PC consumers, and thus any technique to increase clock frequency is strongly required. Pipelining [3, 4, 17] is one of the techniques realizing the high speed circuits and can improve the throughput of a function. However, it also increases the latency of the function, especially for resolving mispredicted branch instructions, and thus processor performance sometimes can not be improved by the technique. Thus, beyond a certain pipeline depth, it becomes difficult to design without timing violations.

One of the possible solutions to avoid timing violations provides some tolerant mechanisms. In other words, we propose to give up meeting timing constraints but to tolerate violations [14]. In other words, in order to simplify timing design of high clock frequency microprocessors, we speculate timing constraints. This philosophy can be applied not only for boosting computing power but also for reducing energy consumption [15]. We call this technique *constructive timing-violation (CTV)*. The CTV exploits the observation that the longest path for an individual operation of a logic circuit is generally much shorter than a critical path of the circuit. Furthermore, it utilizes the fact that input signals which decide the critical path are limited to a few variations. In other words, timing violations rarely occur even if the timing constraints on the critical paths are not satisfied.

The rest of this paper is organized as follows. Section 2 introduces an example of the tolerant mechanism utilized in the CTV and their applications to high clock frequency. Section 3 evaluates the potential of the CTV and presents simulation results. Section 4 presents a case study of the CTV by implementing Carry Select Adder (CSLA). Section 5 surveys related work. Finally, Section 6 presents our conclusions.

2 Constructive Timing Violation

Our example is based on a kind of parallelism, that is space redundancy. An element of circuits consists of a main part and checker parts. The main part is responsible for performance —high throughput and low latency—, but it could suffer timing violations. The checker parts, which are timing violation free, support the main part and revert processor state to a safe point where a timing violation is detected. The main and checker parts are equivalent in design, but we distribute different clocks to them. To the main part, we provide clock frequency which is higher than that decided by the critical path, since it is expected that typical delay of the circuit is less than critical delay and that timing violations rarely occur [6]. On the other hand, since the checker parts are used for detecting the timing violations, they work at the frequency which meets the critical delay. In order to maintain throughput, the checker parts are duplicated if necessary. Please note that the main part is essential for maintaining low latency and that the checker parts only maintain throughput. If there are serious dependences between instructions, high throughput also can not be maintained without the main part.

This design technique exploits the fact that the longest path for an individual operation of a logic circuit is generally much shorter than a critical path of the circuit. Furthermore, it utilizes the fact that input signals which decide the critical path are limited to a few variations. Considering the characteristics of logic circuits and their critical paths, the circuits could be designed as the longest path decided by most of the operations for a function is shorter than an expected cycle time. Please note that this technique is applicable to any combinational logics including datapaths and control logics.

We show an example. Figure 1 depicts an ALU which utilizes the technique. While it is true that in many cases caches are critical and decide clock frequency of a microprocessor, it is reported that adders and instruction scheduling mechanisms become bottlenecks limiting the increase in clock frequency [7, 11]. For example, Compaq's Alpha 21264 processor splits execution pipes into two clusters to meet timing constraints, while it has the data cache that operates at twice the frequency of the processor clock [5]. Thus, we believe ALU is suitable for explaining our proposal because of its simplicity. The shaded box shows additional circuits for the checker parts. It is assumed that the clock frequency decided by the critical delay is f_L. Now we would like to increase the frequency up to f_H, where $f_L \leq f_H < 2 * f_L$. First, the ALU is duplicated

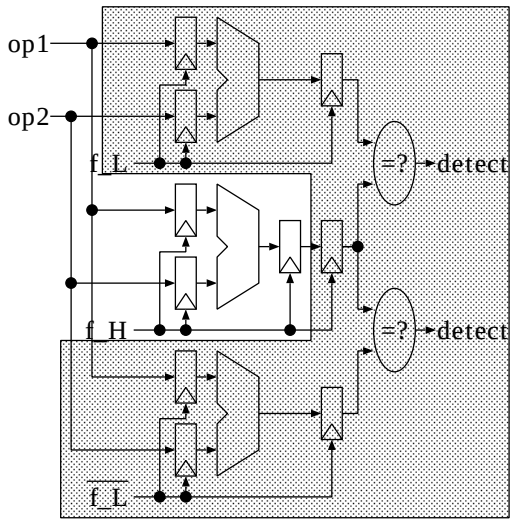

Figure 1. ALU utilizing proposed technique

by three times. One ALU, which we call main ALU, works at f_H and the remaining two ALUs, which we call checker ALUs, work at f_L. Thus, timing violations do not occur in the checker ALUs and thus the checker ALUs are used for verifying the operation of the main ALU with maintaining throughput of the original ALU. Please remember that the main ALU is essential for maintaining low latency and that the checker ALUs only maintain throughput. If there are serious dependences between instructions, high throughput can not be maintained without the main ALU.

It should be explained again that this approach is applicable not only to datapath such as ALU but also to any combinational logics. One of the applications of this approach for control path is the logic detecting data dependences between instructions. This logic is on one of the critical paths in in-order issue microprocessors. Assuming this logic is asserted and is critical only when there are dependences, the CTV mechanism can relief timing constraints so as not to cause timing violations only if there are not any dependences between instructions.

Clock signals distributed to the ALUs are shown in Figure 2. Since the clock signals of the checker ALUs are complementary with each other, they work alternatively to verify the main ALU. Figure 2 explains how two consecutive operations start and are verified. The verification is based on comparing two execution results from the main ALU and a corresponding one of the checker ALUs. If they do not match, a timing violation is detected. In such cases, any recovery action should be initiated. Since the comparators should be violation free, they will work at slower clock frequency

Table 1. Processor configuration

Fetch Width	4 instructions
Branch Predictor	512-set, 4-way set-associative BTB, 2048-entry bimodal predictor, updated in commit stage, 8-entry return address stack, 3-cycle miss penalty
Insn. Windows	16-entry instruction queue, 8-entry load/store queue
Issue Width	4 instructions
Commit Width	4 instructions
Functional Units	4 iALU's, 1 iMUL/DIV, 2 Ld/St's, 4 fALU's, 1 fMUL/DIV
Latency(total/issue)	iALU 1/1, iMUL 3/1, iDIV 20/19, Ld/St 2/1, fADD 2/1, fMUL 4/1, fDIV 12/12
Register Files	32 32-bit fixed point registers, 32 32-bit floating point registers
Insn. Cache	16K direct-mapped, 32-byte blocks, 6-cycle miss penalty
Data Cache	16K 4-way set-associative, 32-byte blocks, 2-port, write-back, non-blocking load, hit under miss, 6-cycle miss penalty
L2 Cache	unified, 256K 4-way set-associative, 64-byte blocks, 48-cycle miss penalty

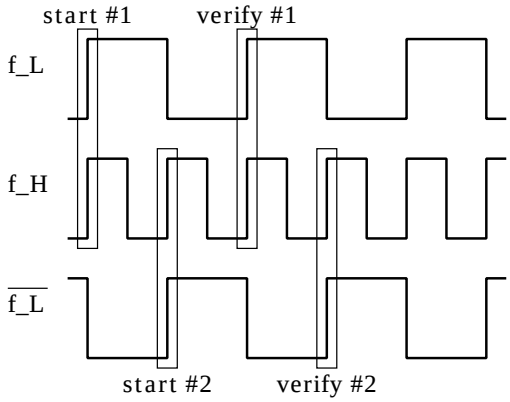

Figure 2. Clock signals

f_L. In order to revert processor state to a safe point where the violation is detected, we need some recovery mechanism for timing violations. We propose to apply the recovery mechanism utilized in data speculation [9, 12, 13] to this purpose. In other words, an instruction that causes a timing violation is regarded as a mispredicted instruction. Thus, we require little hardware overhead in the recovery mechanism for timing violations, since its key component will be implemented in future microprocessors. Please note that the correct value is provided by the checker ALUs, and thus instruction retry is successful for recovery from timing violations. That is, when a violation is detected, it is enough to re-execute instructions following the violating instruction. When a timing violation is detected, the microprocessor selectively re-issue instructions dependent upon the violating instruction. From these considerations, it is possible to detect timing violations and to tolerate them.

3 Potential of CTV

The goal of this section is not to convince that the CTV is practical, but to present its potential on improving computing-power. First, we describe our evaluation environment. After that, we will show simulation results.

3.1 Evaluation environment

We implemented a cycle-by-cycle simulator using SimpleScalar/Alpha tool set (ver.3.0a) [2]. The baseline model is an out-of-order execution superscalar processor based on the register update unit [16], and its configuration is summarized in Table 1.

The SPEC2000 CINT benchmark suite is used for this study. Table 2 lists the benchmarks and the input sets. We use the object files provided by University of Michigan. They were compiled by DEC C V5.9-008 on Digital UNIX V4.0 (Rev.1229). For each program, 1 billion instructions are skipped before actual simulation begins. Each program is executed to completion or for 100 million instructions. We do not count nop instructions.

When we use one main part and two checker parts, the clock frequencies f_L and f_H should satisfy the fol-

Table 2. Benchmark programs

program	input set
164.gzip	input.compressed
175.vpr	net.in arch.in
176.gcc	cccp.i
197.parser	test.in
255.vortex	lendian.raw
256.bzip2	input.random

lowing condition.

$$f_L \leq f_H < 2 * f_L$$

In this evaluation, we choose the clock frequencies of $f_H = 1.5 * f_L$ and f_L for the main and checker parts respectively. In order to evaluate how timing violations affect processor performance, we make them occur randomly. We vary probability that timing violations occur between 0 and 30% of operations, and measure processor performance. We call the probability *fault probability*. Evaluation on energy efficiency can be found in [15].

3.2 Results

In this section, we present preliminary results using the approach described in Section 2. Please note that the technique should be applied to every element which probably violates timing constraints. For measuring performance, we use the committed instructions per second (IPS). Only useful instructions are considered for counting the IPS. In other words, nop instructions or those flushed by branch mispredictions or timing violations are not included when we calculate the IPS.

Figure 3 presents processor performance when the CTV is utilized. Performance of the evaluated model is normalized by that of the baseline model, which is explained in Table 1. Thus, if the performance in figure marks larger than 100%, we can find that processor performance is improved. Figure 1 shows that the proposed technique improves processor performance for all cases even when the fault probability equals 30%. Especially in the case of `255.vortex`, processor performance is improved by 40% even if the fault probability is 30%. In other words, boosting clock frequency by 50% improves processor performance by 40%. The efficiency of boosting clock frequency is 80%. This is more efficient than deep pipelining [4, 17], because deep pipelining increases branch misprediction penalty. From the simulation results, it is found that using the CTV it is possible to increase clock frequency with

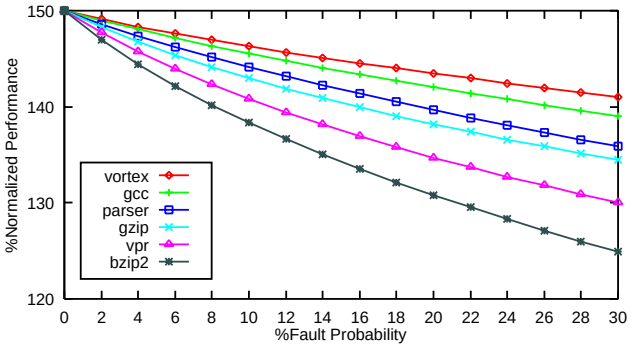

Figure 3. Processor performance

maintaining throughput and latency, and that the high clock frequency contribute to processor performance directly.

4 A Case Study: CSLA

This section presents a case study on evaluating practicability of the CTV. In order to model the relationship between boosted clock frequency and timing error rate, we implement CSLA [10]. Matsuo et al. evaluated the similar technique as the CTV on Carry Lookahead Adder [8].

4.1 Evaluation environment

We implement our CSLA using Verilog-HDL due to the lack of transistor-level process technology information. It is shown in Figure 4. After verifying the correctness of its function by logic simulation, we logic-synthesize it using Synopsis DesignCompiler. The synthesized CSLA includes estimated gate and wire delay based on a standard ASIC library.

In order to detect timing violation, we simulate two CSLAs simultaneously on Cadence Verilog-XL, as shown in Figure 5. One is the upper CSLA with considering delay, which is the gate-level circuit synthesized by DesignCompiler. The other is the lower CSLA without considering delay, which is the RTL description that we implemented using Verilog-HDL. As increasing clock frequency distributed to the registers, the comparator signals there is a difference between results from the two CSLAs.

In order to perform logic simulation, we make test vectors from functional simulation using SPEC2000 CINT benchmark listed in Table 2. Due to large simulation time, we restrict test vectors only 40 thousand patterns for each program.

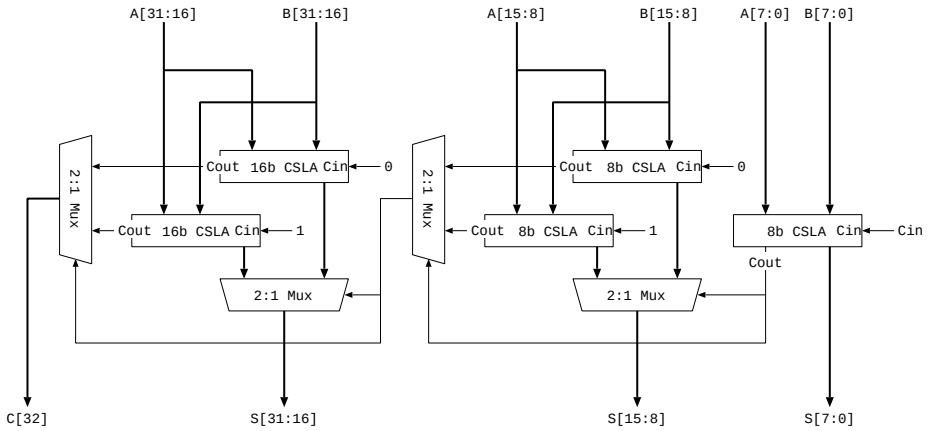

Figure 4. Carry select adder

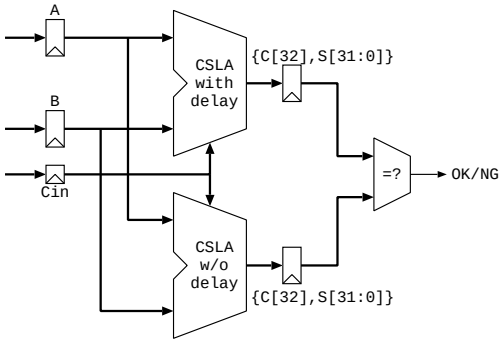

Figure 5. Evaluation circuit

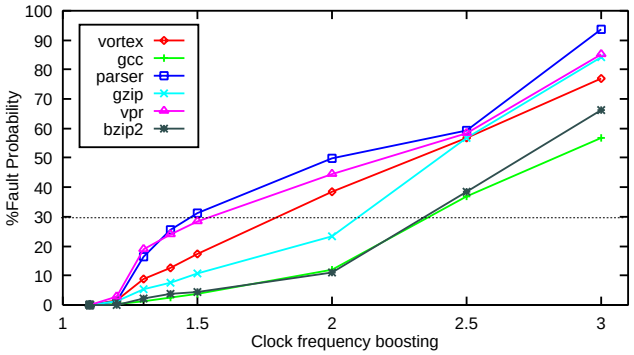

Figure 6. Fault probability

4.2 Results

Figure 6 shows logic simulation results, when we boosted processor clock frequency by the order of 1.1 – 3.0. The horizontal line indicates the boosting ratio and the vertical line indicates the fault probability. We can find that fault probability is less than approximately 30% when clock frequency is 1.5 times faster than that satisfies timing constraints due to critical paths. For example, in the case of `176.gcc`, the fault probability is 3.9% when $f_H = 1.5 * f_L$. Figure 3 shows that processor performance is improved by 48% when the fault probability is 4%. That means clock improvement almost contributes to processor performance. This is unachievable for deep pipelining.

Figure 7 summarizes the fault probability when $f_H = 1.5 * f_L$. In Section 3, we have already evaluated how timing violations affect processor performance, and found that 30% of the fault probability did not have severe impact on performance as shown in Figure 3. Thus, processors may tolerate the measured 30% of the fault probability, and it is confirmed that the CTV technique is effective on boosting processor performance and on simplifying timing design.

5 Related Work

Kondo et al. [6] propose Variable Latency Pipeline (VLP) structure for integer ALUs. Using properly two kinds of circuits according to the longest path of the circuits for each operation, the effective execution latency can be almost one cycle while its critical path is longer than one cycle. Our proposal is strongly influenced by the VLP. However, our proposal is applicable not only to ALUs but also to any combinational logics. Using SPEC92 CINT benchmark suite, the usefulness of the VLP is evaluated on a platform of an in-order execution single-issue processor, but it is not clear for dynamically-scheduled multiple-issue processors. In contrast, we made our evaluation on the out-of-order execution 4-way superscalar processor.

DIVA [1] is an example of a fault-tolerant microprocessor based on space redundancy. A simple checker

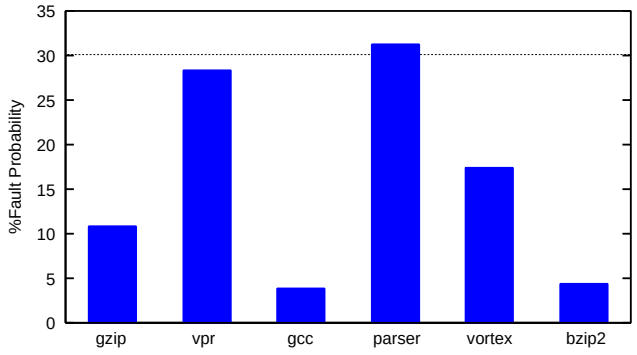

Figure 7. Fault probability ($f_H = 1.5 * f_L$)

processor is used for dynamically verifying committed instructions. Any hardware faults are corrected using recovery mechanism for incorrect branch predictions. One of the problems on DIVA is that it requires additional ports for register files and caches in order for the checker processor to share processor contexts with the main processor. This increases design complexity and circuit delay of its main processor.

6 Conclusion

In this paper, we proposed the constructive timing-violation (CTV) for improving processor performance. Increasing clock frequency causes timing violations, resulting in logic errors. However, if any tolerant mechanism for the timing violations is provided, these errors can be avoided. We explained the example mechanism which is based on space redundancy and modern speculative execution. Every timing violation is detected by the checker circuits and can be recovered by the misprediction recovery mechanism.

Using cycle-by-cycle simulations, we have found that the CTV efficiently improves processor performance. When clock frequency is boosted by the factor of 1.5, the CTV improves processor performance for all cases even when the fault probability equals 30%. The boosting clock frequency by 50% improves processor performance by up to 40%.

In order to evaluate practicability of the CTV, we implemented CSLA and investigate its fault probability. We found that fault probability is less than approximately 30% when clock frequency is 1.5 times faster than that satisfies timing constraints due to critical paths. Because 30% of the fault probability did not have severe impact on performance, processors may tolerate the measured 30% of the fault probability, and it is confirmed that the CTV technique is effective on boosting processor performance and on simplifying timing design.

References

[1] T.M.Austin, "DIVA: a reliable substrate for deep submicron microarchitecture design," 32nd International Symposium on Microarchitecture, 1999.

[2] D.Burger and T.M.Austin, "The SimpleScalar tool set, version 2.0," ACM SIGARCH Computer Architecture News, vol.25, no.3, 1997.

[3] A.Hartstein and T.R.Puzak, "The optimum pipeline depth for a microprocessor," 29th International Symposium on Computer Architecture, 2002.

[4] M. S. Hrishikesh, N. P. Jouppi, K. I. Farkas, D. Burger, S. W. Keckler, and P. Shivakumar, "The optimal useful logic depth per pipeline stage is 6-8 FO4," 29th International Symposium on Computer Architecture, 2002.

[5] R.E.Kessler, E.J.McLellan, and D.A.Webb, "The Alpha 21264 microprocessor architecture," International Conference on Computer Design, 1998.

[6] Y.Kondo, N.Ikumi, K.Ueno, J.Mori, and M.Hirano, "An early-completion-detecting ALU for a 1GHz 64b datapath," International Solid State Circuit Conference, 1997.

[7] T.Liu and S-L.Lu, "Performance improvement with circuit-level speculation," 33rd International Symposium on Microarchitecture, 2000.

[8] T. Matsuo, T. Fujikawa, K. Metsugu, and K. Murakami, "Dependable pipelining: microarchitecture for the multi-GHz generation," Technical Report of IEICE, DC2002-20, 2002 (in Japanese).

[9] E.Morancho, J.M.Llaberia, and A.Olive, "Recovery mechanism for latency misprediction," 10th International Conference on Parallel Architectures and Compilation Techniques, 2001.

[10] V.G.Oklobdzija, "High-speed VLSI arithmetic units: adders and multipliers," in A. Chandrakasan, W. J. Bowhill, and F. Fox, "Design of high-performance microprocessor circuits," IEEE Press, 2001.

[11] S. Palacharla, N. P. Jouppi, and J. E. Smith, "Complexity-effective superscalar processors," 24th International Symposium on Computer Architecture, 1997.

[12] E.Rotenberg, Q.Jacobson, Y.Sazeides, and J.Smith, "Trace processors," 30th International Symposium on Microarchitecture, 1997.

[13] T.Sato, "Analyzing overhead of reissued instructions on data speculative processors," Workshop on Performance Analysis and its Impact on Design, 1998.

[14] T.Sato and I.Arita, "Give up meeting timing constraints, but tolerate violations," Cool Chips IV, 2001.

[15] T.Sato and I.Arita, "Constructive timing violation for improving energy efficiency," 2nd Workshop on Compilers and Operating Systems for Low Power, 2001.

[16] G. S. Sohi, "Instruction issue logic for high-performance, interruptible, multiple functional unit, pipelined computers," IEEE Transactions on Computers, vol.39, no.3, 1990.

[17] E.Sprangle and D.Carmean, "Increasing processor performance by implementing deeper pipelines," 29th International Symposium on Computer Architecture, 2002.