Ramp Up/Down Floating Point Unit To Reduce Inductive Noise

Zhenyu Tang*, Norman Chang, Shen Lin, Weize Xie, Sam Nakagawa, and Lei He*
Hewlett-Packard Laboratories, Palo Alto, CA 94306
*ECE Dept., Univ. of Wisconsin, Madison, WI 53706

Abstract— **Because the inductive noise Ldi/dt is induced by the power change and can have disastrous impact on the timing and reliability of the system, high-performance CPU designs are more concerned with the step power reduction instead of the average power reduction. The step power is defined as the power difference between the previous and present clock cycles, and represents the Ldi/dt noise at the microarchitecture level. Two mechanisms at the microarchitecture level are proposed in this paper to reduce the step power of the floating point unit (FPU), as FPU is the potential "hot" spot of Ldi/dt noise. The two mechanisms, ramping up and ramping down FPU based on instruction fetch queue (IFQ) scanning and $PC + N$ instruction prediction, can meet any specific step power constraint. We implement and evaluate the two mechanisms using a performance and power simulator based on the SimpleScalar toolset. Experiments using SPEC95 benchmarks show that our method reduces the performance loss by a factor of four when compared to a recent work.**

I. Introduction

Because of the growing transistor budget, increasing clock frequency and wider datapath width in the modern processor design, there is an ever-growing current to charge/discharge the power/ground buses in a short time [1], [2]. When the current passes through the wire inductance (L) associated with power or ground rails, the voltage glitch is induced and is proportional to Ldi/dt, where di/dt is the current changing rate. Therefore, the power surge is also known as Ldi/dt noise. Further, many power-efficient microarchitecture techniques involve selectively throttling, or clock gating certain functional units or parts of functional units [3], [4], [5], [6], [7], [8]. Dynamic throttling techniques may lead to an even larger surge current.

A large surge current may reduce the chip reliability, and cause timing and logic errors, i.e., a circuit may switch at the wrong time and latch the wrong value. Dealing with a large surge current needs an elaborate power distribution and contributes to higher design and manufacturing costs. In this paper, we define the *step power* as the power difference between the previous and present clock cycles. We use the step power as a figure of merit for power surges at the microarchitecture level, and study how to reduce the step power by ramping up and ramping down functional units for high-performance processors.

We use the floating point unit (FPU) to illustrate our ideas. The FPU consumes 10-20% power of the whole system. Its step power has a significant impact on power delivery and signal integrity in processor design. The FPU is turned on or off *immediately* in most previous research on the dynamic throttling, and results in a large step power. Recently, Tiwari *et al* [9], [2] proposed to prolong the switch on/off time by inserting "waking up" and "going to sleep" time between the *on* and *off* state. However, every time the pipeline is forced to stall several clock cycles before an inactive resource becomes available. This may lead to a large performance penalty.

In this paper, we proposed two new mechanisms to ramp up/down (turn on/off gradually) the FPU based on either the *instruction fetch queue (IFQ) scanning* or the *PC+N instruction prediction* to meet the step power constraint specified by the designer. The main difference between our work and Tiwari's is that we *predict* the instruction in advance whether the resource is required. This will enable a request signal to be placed early enough to ramp up the inactive FPU *gradually* and make it ready for use in time. We implement and evaluate our two mechanisms using a performance and power simulator based on the SimpleScalar toolset. Compared to [9], [2], we can reduce the performance loss by a factor of 4 on average over SPEC95 benchmarks.

The paper is organized as follows. Section 2 describes our two step power reduction mechanisms in detail. Section 3 presents the quantitative study of the step power and performance impact of the two mechanisms. Section 4 discusses the possible implementation method of the two mechanisms, and section 5 concludes this paper.

II. The step power reduction mechanisms

A. Overview

As the first step towards the power reduction techniques, we have implemented an accurate microarchitecture level power estimation tool based on the extended SimpleScalar toolset, where the SimpleScalar architecture [10] is divided into thirty-two functional blocks, and activity monitors are inserted for these functional blocks [11], [12]. We have developed a power hook as an interface between the extended SimpleScalar toolset and the RTL power models of functional blocks. After reading the system configuration and user specified RTL power information coming from the real design data or RTL power estimation tool [13], [14], [15], [16], [17], the activities and the corresponding power information of these functional blocks are collected in every clock cycle. Our resulting SimpleScalar toolset is able to simulate the performance, average power and step power

Part of this work was performed while Mr. Tang was an intern with HP in the 2000 summer. Address comments to nchang@hpl.hp.com and lhe@ece.wisc.edu.

for every functional block and the whole system for given benchmark programs. All our power reduction techniques are implemented and tested on this toolset.

The conventional floating point unit (FPU) design only has two states: *inactive* state and *active state* (see Figure 1(a)). When there are floating point instructions executed, the FPU is in the active state and consumes the active power (P_a). On the other hand, FPU has no activity in the inactive state and dissipates the leakage power (P_i), about 10% of the active power (P_a) in the present process technology. When any floating point instruction gets into the FPU, the FPU will jump up from the inactive state to the active state in one clock cycle and has a step power of ($P_a - P_i$) (see Figure 1(a)). If we assume that the inactive power (leakage power) is 10% of the active power, the step power of FPU will reach $0.9P_a$ and may translate into a large Ldi/dt noise.

Essentially, Figure 1 (b) illustrates the technique used in Tiwari's work [9], [2]. Stall cycles will be inserted to power up the functional units gradually *every time* when the inactive resources are needed and may lead to a big loss of the performance. However, our work *predicts* the occurrence of the floating point instructions and prepares the FPU in advance to reduce this performance penalty. In both Tiwari's and our approaches, the FPU will be powered down gradually to save power consumption, when it is not used for certain clock cycles.

We introduce several *artificial workload states* in our approach. The relationship of the inactive state, artificial workload states, and active state of FPU is illustrated in Figure 1 (c). The FPU consumes power P_w^i, i=1,2,...,n, and $P_w^n > P_w^{n-1} > ... > p_w^1$ if there are n artificial workload states. We assume that the power difference between adjacent power states is uniform for the simplicity of presentation. A special power state, which is only one step below the active state, is called *subactive state* and dissipates P_s power. After a floating point instruction is predicted, the FPU will ramp up and stay in the subactive state. The FPU enters the active state when the instruction gets executed. In summary, $P_w^0 = P_i$, $P_w^n = P_s$ and $P_w^{n+1} = P_a$.

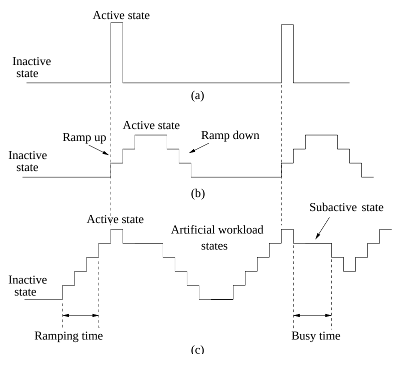

Fig. 1. The relationship of states.

B. Ramp up/down FPU based on the IFQ scanning

The SimpleScalar is a five-stage superscalar architecture. There are two intermediate stages between the instruction fetch (IF) and execution (EXE) stages. We can scan the fetched instructions in the instruction fetch queue (IFQ) of the IF stage every clock cycle. We call this mechanism *IFQ scanning*. If there exist floating point instructions, a request signal will be sent to the EXE stage directly to ramp up the FPU from the inactive state to the subactive state within *prediction time* by adding artificial workload gradually. Here, the prediction time is two cycles between IF and EXE stage. If the floating point instruction really gets into the FPU in EXE stage, the FPU will switch from the subactive state to the active state. Otherwise, FPU will be ramped down to the inactive state after the *busy time*, which is a user defined time to keep FPU in the subactive state. If one floating point instruction appears during the busy time or ramp down time, to reduce the performance penalty, the FPU will ramp up immediately without reaching the inactive state as shown in Figure 1(c).

A prolonged busy time helps to exploit the spatial and temporal locality of operations. If it is set to zero, the FPU will be ramped down immediately after it reaches the subactive state. This will introduce larger performance penalty also due to the following observation: the floating point instruction may be executed out-of-order and cannot get into the FPU within the prediction time. On the other hand, the infinite busy time keeps the FPU in the subactive state and never powers it down. It may increase the performance, but the average power dissipation of FPU will increase a lot since the FPU always consumes the subactive power even in the idle state.

C. Ramp up/down FPU based on instruction prediction

Ramp up/down FPU based on instruction prediction is a more general mechanism to reduce the step power. The main idea is to prefetch one more instruction per cycle from the I-cache and predict whether the FPU is required by this instruction (IFQ scanning is clearly a simple case of this mechanism.) This will help FPU to ramp up gradually in advance and make it available for use in time. Our current implementation is to scan the instruction with address $PC + N$, where PC is the current program counter and N is a user decided parameter. We will ramp up/down the FPU based on the prediction of this instruction as we do in the first method. In this way, we can have $N + 2$ cycles (still including the two cycles between IF and EXE stages) to ramp up/down FPU with a further reduced step power. We define this $N + 2$ as *prediction time*, which can also be viewed as the *ramping time* for the FPU to power on/off gradually between the inactive and the subactive states shown in Figure 1 (c).

Further, there is one power step between the subactive state and active state. Therefore, there are *prediction time*+1 $= N + 3$ power steps between the inactive and active states. We define the step power in this paper as follows:

$$\begin{aligned} \textit{step power} &= (P_a - P_i)/(\textit{prediction time} + 1) \quad (1) \\ &= (P_a - P_i)/(N + 3) \end{aligned}$$

For example, if P_i is $0.1P_a$ and $N = 6$, we can achieve a step power of $0.1P_a$, with an 88.9% reduction compared to the conventional step power of $0.9P_a$. As the IFQ scanning is a simple case of the $PC + N$ instruction prediction with $N = 0$, the FPU in this case has a step power of $0.3P_a$.

Because the SimpleScalar is an out-of-order execution superscalar processor, the predicted instructions may stall in the dispatch stage due to data or resource hazard. This will cause the predicted instructions to be executed at a different time. Extra stall cycles have to be inserted until the FPU reaches the subactive state and becomes available again, which will introduce performance penalty. On the other hand, when $PC + N$ instruction prediction is used, there may be branch instructions among these N instructions and branch misprediction may happen. We currently assume that all the branch instructions are *not* taken, therefore we do not need any extra circuit and keep the cost minimum. Nevertheless, we can utilize other existing branch prediction techniques [18], [19] to reduce misprediction and achieve better results, but with a higher hardware overhead.

We proceed to present the quantitative results on performance and step power, using the SPEC95 FP benchmark programs in the next section.

III. The Implementation Method

We summarize the ramping up/down algorithm based on the $NP + N$ prediction in Figure 2. To ramp up/down the FPU based on instruction prediction, one more instruction will be fetched and predecoded in the IF stage for every clock cycle. In this case, a small predecode and control logic is needed. Two counters *count_fpu_busytime* and *count_fpu_ramptime* are used to count up/down the busy time and prediction time respectively. A logic signal *signal_fpu_ramp* will be used to indicate the state of ramping up or ramping down.

As shown in Figure 2, when the floating point instruction is detected in the instruction fetch stage, *signal_fpu_ramp* will be set to *ramp_up* and sent to the scheduler stage. The counter *count_fpu_busytime* will be reset to 0. If there is no floating point instruction executed by the FPU, *count_fpu_busytime* will start counting. When it reaches busy time, *signal_fpu_ramp* is changed to *ramp_down* and *count_fpu_busytime* will be reset again.

The scheduler stage will keep checking the status of signal_fpu_ramp. If it is set to *ramp_up*, *count_fpu_ramptime* will be used to ramp up FPU from the inactive state to the subactive state within prediction time by increasing the workload of FPU gradually. The FPU is ready for execution in the subactive state. On the other hand, if signal_fpu_ramp is set to *ramp_down*, *count_fpu_ramptime* will be decremented to ramp down FPU gradually and FPU is not available then.

In the execution stage of simplescalar, floating point instruction is issued to FPU only when the FPU is available, which is decided in the scheduler stage. Otherwise, floating point instruction has to stall and waits for the FPU ramping up. If FPU is in the inactive or ramping down state and a new floating point instruction appears, the FPU starts to ramp up immediately to reduce performance penalty.

In terms of circuit implementation, both the clock network and FPU can be partitioned into subcircuits. One subcircuit will be enabled or disabled per cycle during ramping up or ramping down via clock gating as discussed in [20], [21].

IV. The Experiment Results

In this section, SPEC95 FP benchmark programs are used to study the performance impacts of the two FPU step power reduction techniques. The performance is presented by instructions per cycle (IPC). We use the performance without any ramping up and down as the base to measure the performance loss, and summarize the system configuration used in our experiment in Table I.

A. Impact of busy time

Figure 3 shows the performance loss in the IFQ scanning mechanism. In this figure, the constant prediction time is two, the constant step power is $0.3P_a$, but the busy time varies from five to fifteen. As expected, the IPC loss is reduced when the busy time is increased, because the FPU has more time to stay in the subactive state and better prepares for the execution of floating point instructions. The IPC loss is less than 2.0% for the FP programs, when busy time is ten clock cycles.

By using the $PC + N$ instruction prediction mechanism with a prediction time of eight, we can achieve the step power of $0.1P_a$ (88.9% reduction compared to the conventional design with only active and inactive power states). The busy time varies from five to fifteen clock cycles in Figure 4. Again, the performance penalty is smaller when the busy time increases. The performance loss is less than 3.0% when the busy time is larger than ten clock cycles. However, the longer FPU stays in the subactive state, the more power it consumes, as the subactive power is much larger than the inactive power. Therefore, there exists a tradeoff between performance penalty and average power reduction. It can be controlled by the user defined busy time. Because the performance loss becomes smooth and less than 3.0% when the busy time is larger than ten cycles, we choose ten clock cycles as the busy time in the remaining of this section.

B. Impact of prediction time

Figure 5 reflects the relationship between performance loss and prediction time. The step power is $0.225P_a$ by setting $N = 1$ and *prediction time* $= 3$. This prediction leads to less than 1.2% performance penalty for all benchmarks. Generally, when the required step power becomes smaller, a bigger N is needed as the FPU needs more

```
The Microarchitectural Level Instruction Prediction Algorithm :
    /* In the INSTRUCTION FETCH stage */
        Prefetch instruction pwr_instr that is N cycles later;
        Predecode this instruction pwr_instr;
        if (pwr_instr is FP instruction) {
            signal_fpu_ramp = ramp_up; //start to ramp up FPU
            reset count_fpu_busytime;
        } else if (FPU is in the subactive state){
            if (count_fpu_busytime == busytime) {
                signal_fpu_ramp = ramp_down; //start to ramp down FPU
                reset count_fpu_busytime;
            } else
                count_fpu_busytime++;
        }
    /* In the SCHEDULER stage */
        if (signal_fpu_ramp == ramp_up) {
            if (FPU reaches the subactive state)
                FPU_is_available = 1;
            else {
                FPU_is_available = 0;
                count_fpu_ramptime++;
            }
        } else if (signal_fpu_ramp == ramp_down) {
            if (count_fpu_ramptime > 0) {
                FPU_is_available = 0;
                count_fpu_ramptime--;
            }
        }
    /* In the EXECUTION stage */
        if (the instruction is floating point instruction)
            if (FPU_is_available)
                issue FP instruction to FPU;
            else {
                stall FP instruction;
                start to ramp up FPU immediately;
            }
        }
```

Fig. 2. Microarchitecture Level Instruction Prediction Algorithm

Functional Unit	number	Latency		
Integer-ALU	4	1		
Integer-MULT/DIV	1	3/20		
Floating Point Adder	4	2		
Floating Point MULT/DIV	1	4/12		
Fetch/decode/issue width	4			
Memory bus width	8			
Cache	nsets	bsize	assoc	Repl
I-L1	512	32	1	LRU
D-L1	128	32	4	LRU
I-TLB	16	4096	4	LRU
D-TLB	32	4096	4	LRU
U-L2	1024	64	4	LRU

TABLE I

System configuration for experiments.

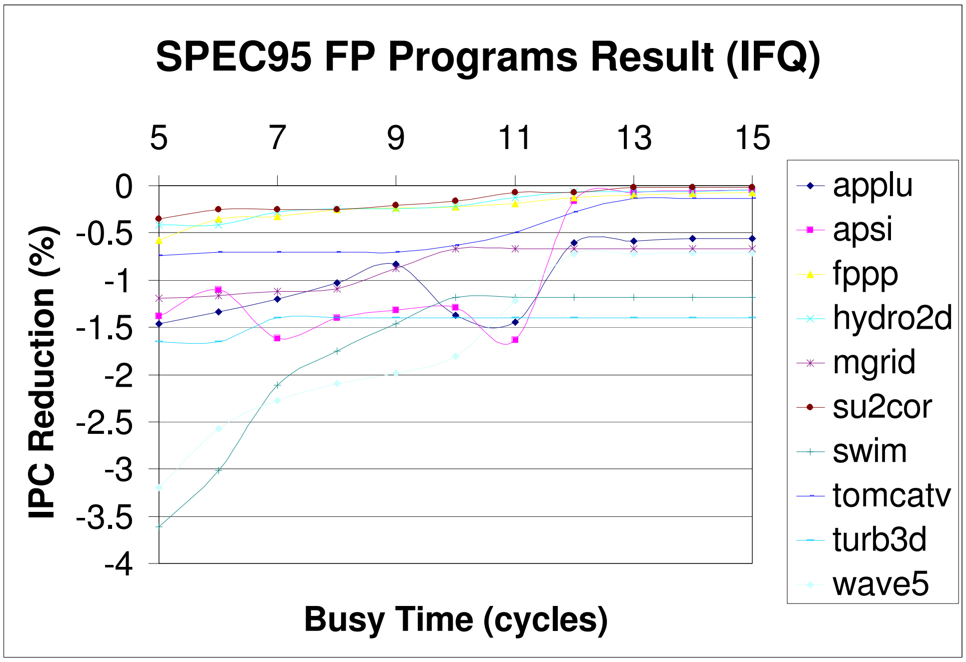

Fig. 3. Performance penalty versus busy time in IFQ scanning with a prediction time of two.

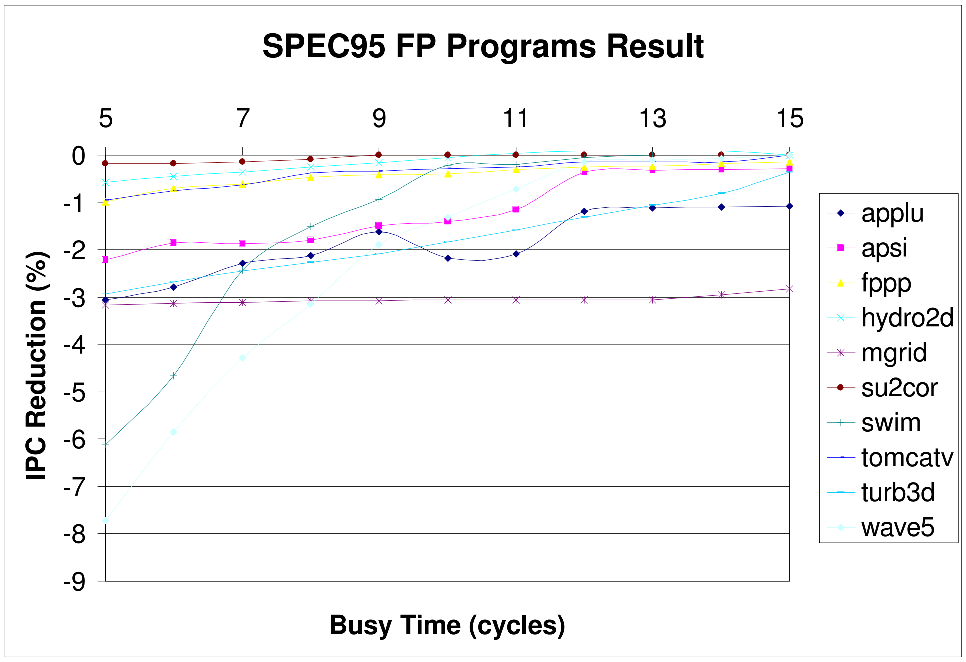

Fig. 4. Performance penalty versus busy time for $PC + N$ instruction prediction. The prediction time is eight.

time to ramp up/down. The potential chance of misprediction will increase, and more performance loss may be induced. For example, if the required step power is $0.1P_a$ as often required in the real design, we can set $N = 6$ and *predicton cycle* $= 8$. The performance penalty is increased but is still relatively small. The performance loss is less than 3.0% for all benchmarks. Clearly, there exists a tradeoff between the performance penalty and step power reduction. The tradeoff can be adjusted by the prediction time. As shown in Figure 5, benchmarks *turb3d* and *swim* are sensitive to the prediction time. It may be due to our branch prediction scheme, and is worthwhile further investigation.

An alternative to achieve $0.1P_a$ without using ramping up (and therefore without performance loss) is to set the inactive power $P_i = 0.9P_a$. Dummy operations are used to keep this level of inactive power when no instruction really needs the FPU. This leads to a huge amount of non-necessary power dissipation. With our prediction technique, the FPU can only consume the leakage power in the inactive state. As the leakage power is about $0.1P_a$ for the current process, we can achieve a factor of nine in terms of power saving in the inactive state. Note that even the leakage power can be saved if the power gating is used to cut off the power supply.

C. Comparison with previous work

In the following, we compared our performance loss with that in Tiwari's work [9], [2], where the pipeline is forced to install in order to ramp up the inactive FPU. In Figures 6 and 7, light-colored bars are performance losses without using prediction based on our own implementation of the approach in [9], [2], and dark-colored bars are performance losses using our IFQ scanning prediction technique in figure 6 and N-cycle instruction prediction technique in figure 7. The busy time is ten for both figures, and the prediction time (same as ramping time) is two and eight in Figures 6 and 7, respectively. One can see that our prediction techniques achieve much better performance. When the prediction time is two, the average performance loss is 1.96% without prediction, but is only 0.90% with prediction. When the prediction time is eight, the average performance loss is 4.65% without prediction, but is only 1.08% with prediction. The prediction reduced the performance loss by a factor of more than 4 in both cases.

V. Conclusions and discussions

Based on an extended SimpleScalar toolset and the SPEC95 benchmark set, a preliminary study has been presented at the microarchitecture level to reduce the inductive noise by ramping up/down the floating point unit. Instruction fetch queue (IFQ) scanning and $PC + N$ instruction prediction have been proposed to reduce the performance loss due to ramping up and ramping down. Our techniques are able to satisfy any given constraint on the step power, and reduce the performance loss by a factor of more than 4 when compared with a recent work without prediction.

We assume that the ramping time is same as the prediction time in this paper. In general, the two times may be different to further reduce the performance loss. It is part of our ongoing work to find the optimal prediction time for the ramping time determined a given step power constraint. We also plan to apply the prediction mechanism to other functional units such as the integer ALU and Level-2 cache. Further, we intend to investigate the power impact due to ramping up and down in the context of the whole system, but not just an individual functional unit as in this paper.

Acknowledgments

The authors would like to thank Dr. George Cai at Intel for providing his enhanced version of SimpleScalar toolset and for his helpful discussions.

References

[1] Y.-S.Chang, S.K.Gupta, and M.A.Breuer, "Analysis of ground bounce in deep sub-micron circuits," in *VLSI Test Symposium, IEEE, IEEE Computer Society Press*, pp. 110–116, 1997.

[2] M. Pant, P. Pant, D. Wills, and V.Tiwari, "An architectural solution for the inductive noise problem due to clock-gating," in *Proc. Int. Symp. on Low Power Electronics and Design*, pp. 255–257, 1999.

[3] S.Manne, A.Klauser, and D.Grunwald, "Pipeline gating: Speculation control for energy reduction," in *International Symposium on Computer Architecture*, 1998.

[4] A.Raghunathan, S.Dey, A.Horak, T.Mudge, and K.Roy, "Low-power system design: Application, architectures, and design methodologies," in *Proc. Design Automation Conf*, 2000.

[5] N.Vijaykrishnan, M.Kandemir, M.J.Irwin, and H.S.Kim, "Energy-driven integrated hardware-software optimization using simplepower," in *International Symposium on Computer Architecture*, pp. 95–106, 2000.

[6] E.Musoll, "Predicting the usefulness of a block result: a micro-architectural technique for high-performance low-power processors," in *32nd Annual International Symposium on Microarchitecture*, November 1999.

[7] E. Macii, M. Pedram, and F. Somenzi, "High-level power modeling, estimation and optimization," in *Proc. Design Automation Conf*, 1997.

[8] Y. Li and J. Henkel, "A framework for estimating and minimization energy dissipation of embedded hw/sw systems," in *Proc. Design Automation Conf*, 1998.

[9] M. Pant, P. Pant, D. Wills, and V. Tiwari, "Inductive noise reduction at the architectural level," in *International Conference on VLSI Design*, pp. 162–167, 2000.

[10] D. Burger and T. Austin, *The simplescalar tool set version 2.0*. University of Wisconsin-Madison, 1997.

[11] G. Z. Cai, K. Chow, T. Nakanishi, J. Hall, and M. Barany, "Multivariate power/performance analysis for high performance mobile microprocessor design," in *Power-Driven Microarchitecture Workshop In Conjunction With ISCA98*, June 1998.

[12] A. Dhodapkar, C. Lim, and G. Cai, "Tem2p2est: A thermal enabled multi-model power/performance estimator," in *Workshop on Power Aware Computer Systems*, Nov 2000.

[13] Q.Wu, Q.Qiu, M.Pedram, and C.Ding, "Cycle-accurate macro-models for rt-level power analysis," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 6, pp. 520–528, December 1998.

[14] H. Mehta, R. Owens, and M. Irwin, "Energy characterization based on clustering," in *Proc. Design Automation Conf*, June June 1996.

[15] F.N.Najm, "A survey of power estimation techniques in VLSI circuits," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 2, pp. 446–455, December 1994.

[16] S. Gupta and F. N. Najm, "Power macromodeling for high level power estimation," in *Proc. Design Automation Conf*, pp. 365–370, June 9-13 1997.

[17] D. Liu and C. Svensson, "Power consumption estimation in

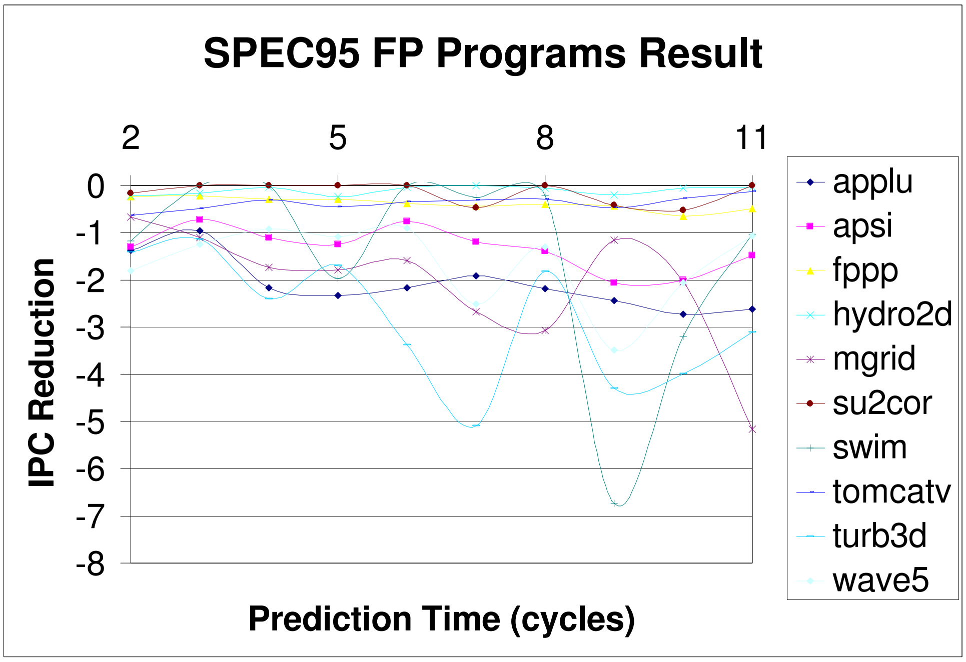

Fig. 5. Performance penalty versus prediction time for $PC + N$ instruction prediction. The busy time is ten cycles. $Prediction\ time = 2$ stands for IFQ scanning.

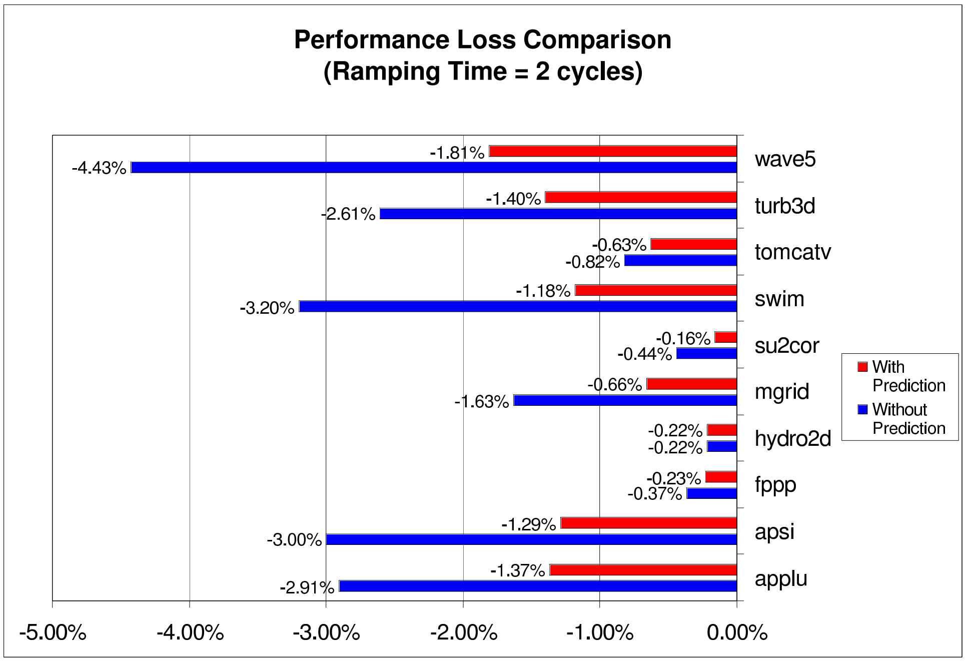

Fig. 6. Performance Loss Comparison between IFQ Scanning prediction and non-prediction for SPEC95 FP programs (ramping time is two cycles).

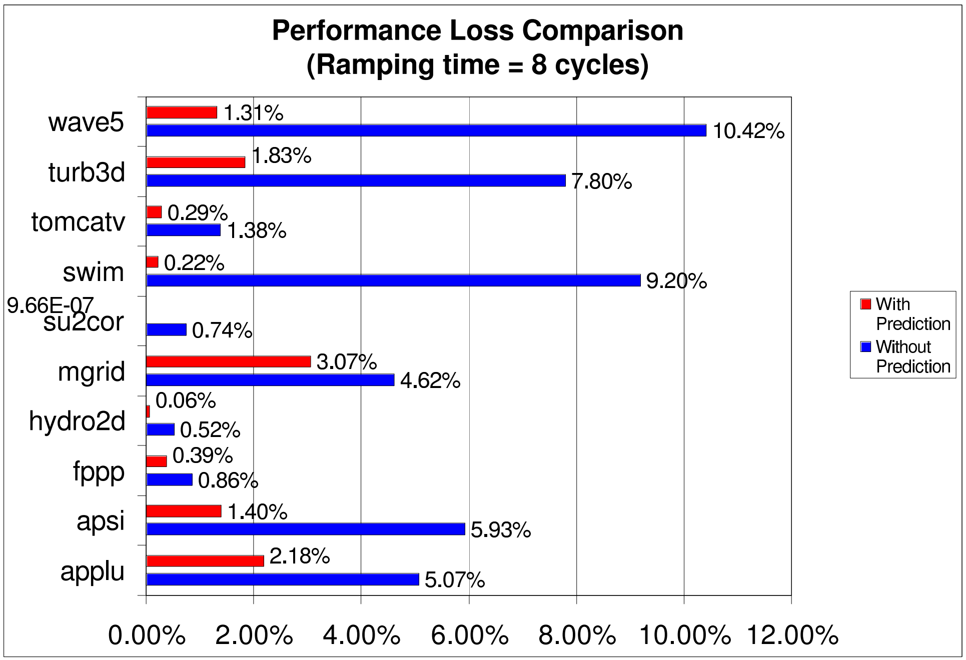

Fig. 7. Performance Loss Comparison between N-cycle instruction prediction and non-prediction for SPEC95 FP programs (ramping time is eight cycles).

CMOS VLSI chips," *IEEE Journal of Solid-state Circuits*, pp. 663–670, June 1994.

[18] J.L.Hennessy and D.A.Patterson, *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publishers, 1996.

[19] J.Fisher and S.Freudenberger, "Predicting conditional branch directions from previous runs of a program," in *Proc. Fifth Conf. on Architectural Support for Programming Languages and Operating Systems, IEEE/ACM*, pp. 85–95, October 1992.

[20] V. Tiwari, D. Singh, S. Rajgopal, and G. Mehta, "Reducing power in high-performance microprocessors," in *Proc. Design Automation Conf*, pp. 732–737, 1998.

[21] E. Macii, M. Poncino, and R. Zafalon, "RTL and gate-level power optimization of digital circuits," in *Proc. IEEE Int. Symp. on Circuits and Systems*, 2000.