VSV: L2-Miss-Driven Variable Supply-Voltage Scaling for Low Power

Hai Li, Chen-Yong Cher, T. N. Vijaykumar, and Kaushik Roy

1285 EE Building, ECE Department, Purdue University
<hl, chenyong, vijay, kaushik>@ecn.purdue.edu

Abstract

Energy-efficient processor design is becoming more and more important with technology scaling and with high performance requirements. Supply-voltage scaling is an efficient way to reduce energy by lowering the operating voltage and the clock frequency of processor simultaneously. We propose a variable supply-voltage scaling (VSV) technique based on the following key observation: upon an L2 miss, the pipeline performs some independent computations but almost always ends up stalling and waiting for data, despite out-of-order issue and other latency-hiding techniques. Therefore, during an L2 miss we scale down the supply voltage of certain sections of the processor in order to reduce power dissipation while it carries on the independent computations at a lower speed. However, operating at a lower speed may degrade performance, if there are sufficient independent computations to overlap with the L2 miss. Similarly, returning to high speed may degrade power savings, if there are multiple outstanding misses and insufficient independent computations to overlap with them. To avoid these problems, we introduce two state machines that track parallelism on-the-fly, and we scale the supply voltage depending on the level of parallelism. We also consider circuit-level complexity concerns which limit VSV to two supply voltages, stability and signal-propagation speed issues which limit how fast VSV may transition between the voltages, and energy overhead factors which disallow supply-voltage scaling of large RAM structures such as caches and register file. Our simulations show that VSV achieves an average of 20.7% total processor power reduction with 2.0% performance degradation in an 8-way, out-of-order-issue processor that implements deterministic clock gating and software prefetching, for those SPEC2K benchmarks that have high L2 miss rates. Averaging across all the benchmarks, VSV reduces total processor power by 7.0% with 0.9% performance degradation.

1. Introduction

Power dissipation is becoming a limiting factor in high-performance processor design as technology scales and device integration level increases. Supply-voltage scaling is emerging as an effective technique for reducing both dynamic power and leakage power. Dynamic power is proportional to $f \cdot C_{load} \cdot V_{DD}^2$, where f is the system clock

frequency, C_{load} is the effective load capacitance, and V_{DD} is the supply voltage. Scaling the supply voltage requires a commensurate reduction in clock frequency because signal propagation delays increase when the supply voltage is scaled down. The maximum clock frequency at which a transistor can operate is proportional to $\frac{(V_{DD} - V_T)^{\alpha}}{V_{DD}}$, where V_T is the transistor threshold voltage, and α is strongly dependant on the mobility degradation of electrons in transistors (with typical value between 1 and 2). Therefore, supply-voltage scaling can reduce dynamic power in the order of V_{DD}^2 and V_{DD}^3. Although supply-voltage scaling also reduces leakage power in the order of V_{DD}^3 and V_{DD}^4 [17], we will focus only on dynamic power in this paper.

We consider a specific flavor of supply-voltage scaling, called variable supply-voltage scaling (VSV), where we vary the supply voltage while a program executes. To ensure VSV's effectiveness, we consider several circuit-level overhead issues: First, we choose two convenient, discrete supply voltages after considering the trade-offs between power saving and design complexity. Second, even with two supply voltages, transitioning from one to the other takes time. Because of circuit stability issues discussed in Section 3.2, this transition cannot be done fast and needs to be of the order of at least a few clock cycles (e.g., 12 cycles for 0.18μm technology and 1GHz clock).

Because lowering the V_{DD} requires reducing the clock frequency, we consider microarchitectural issues to ensure that VSV achieves power reduction without inordinate performance degradation. A fundamental observation is that if the clock speed is reduced whenever the processor is executing non-critical code, then it is conceivable that performance will not be degraded. Fortunately, it is simple to use cache misses as triggers to flag non-critical processing: Upon a cache miss, the processor executes only the few instructions that are independent of the miss and often ends up stalling, despite out-of-order issue and other latency-hiding techniques. Therefore, we propose to lower the V_{DD} on cache misses. Those few independent instructions are executed slower at the lower V_{DD}, but no significant performance loss would ensue as long as the instructions are completed before the miss returns. Because V_{DD}-transition times are usually of the same order as L2-hit latencies (e.g., 12 cycles for the Alpha 21264), V_{DD} should not be lowered for L1 misses that are L2 hits. Consequently, we propose to lower the V_{DD} only on L2

misses, which are long enough (e.g., 100 cycles) to amortize the transition time overhead. Thus, we scale down the V_{DD} of certain processor sections to achieve power reduction, but allow the processor to continue processing the instructions that are independent of the L2 misses, albeit at a reduced speed.

Depending upon the program's degree of instruction-level parallelism (ILP), the processor may find many independent instructions to overlap with L2 misses. If the V_{DD} is lowered in such cases, significant performance loss would ensue. To avoid the loss, VSV employs a novel mechanism using state machines that estimate the degree of ILP by monitoring the instruction-issue rate. If soon after an L2 miss the issue rate is high (defined by some thresholds), VSV does not scale down the V_{DD}. When one of many outstanding L2 misses returns and results in a high issue rate in low V_{DD}, indicating that there are instructions independent of the outstanding misses, VSV switches back to high V_{DD}.

In CMOS, varying the V_{DD} changes the amount of charge held in the internal nodes of circuits – both combinational logic and RAM structures. Because RAM structures contain numerous (of the order of tens or hundreds of thousands) cells, the energy overhead of changing the charge held in each of the cells needs to be amortized by the energy savings achieved by accessing the structures in low V_{DD}. Because the number of cells accessed is a small fraction of the total number of cells in a structure, many low-V_{DD} accesses are needed to make up for the energy overhead of one V_{DD} transition. Unfortunately, accesses to these structures within a typical L2 miss latency are too few to achieve this amortization, as we show in Section 3.6. In combinational logic, however, this amortization is achieved because the number of nodes activated in one low-V_{DD} operation is of the same magnitude as the number of nodes whose charge is changed in one V_{DD} transition. Therefore, we scale the V_{DD} of the pipeline and not of the register file and the caches. We use voltage level converters to handle the cases where the pipeline uses low V_{DD} and the RAM structures use high V_{DD}.

While [18] explores supply-voltage scaling, the paper states that it makes simplistic assumptions of varying the V_{DD} without any power or performance overhead. In contrast, VSV makes the following contributions:

- At the circuit level, VSV takes into consideration the design simplicity of two levels of V_{DD}, the V_{DD}-transition time overhead, and the energy overhead of scaling RAM structures' supply voltages.
- At the architecture level, VSV uses L2 misses and the novel instruction-issue-rate monitoring mechanism as triggers for V_{DD} transitions.

We use Wattch [7] and the SPEC2K suite [8] to simulate an 8-way, out-of-order-issue superscalar for our experiments. Our main results are:

- Most modern processors use clock gating and software prefetching. If the pipeline ends up stalling upon an L2

miss, the power of the unused circuits would be reduced already by clock gating, reducing VSV's opportunity. However, VSV has at least two advantages over clock gating: (1) clock gating cannot reduce power of used circuits while VSV can, and (2) clock gating cannot gate all unused circuits if the clock gate signal's timing is too tight [10]. Prefetching reduces cache misses, directly limiting VSV's opportunity. However, prefetching does not completely eliminate L2 misses, and processors do stall even when aggressive prefetching is used. Consequently, VSV retains enough opportunity.

- In a processor that uses clock gating [10] and software prefetching, our simulations show 20.7% average improvement in total processor power consumption with 2.0% performance degradation for the benchmarks with high L2 miss rates (> 4 misses per 1,000 instructions). The average processor power reduction is 7% over all the SPEC2K benchmarks.
- To stress-test VSV, we consider the recently-proposed Time-Keeping prefetching [9]. We observe that VSV achieves 12.1% average power saving and 2.1% performance degradation for benchmarks with high L2 miss rates, even after employing Time-Keeping on top of clock gating and software prefetching.

The remaining sections are organized as follows. Section 2 presents the related work on voltage scaling. Section 3 discusses the circuit-level issues of VSV. Section 4 describes the implementation details at the architectural level. Section 5 explains the experimental methodology. Section 6 presents the results and shows the impacts of prefetching. Section 7 concludes the paper.

2. Related Work

Many researchers have addressed OS-level or compiler-level supply-voltage scaling techniques. Burd *et. al.* [1] demonstrate a complete embedded system, called the lpARM processor, which includes an ARM8 core with 16KB cache. Pouwelse *et. al.* [2] build a system with Strong ARM processor whose speed can be varied along with its input voltage. Hsu *et. al.* [3] propose a compiler-directed performance model to determine an efficient CPU slow-down factor for memory-bound loop computations. These techniques have at least two common characteristics: (1) these techniques operate at process-level granularity; and (2) to change the clock speed automatically with V_{DD}, these techniques vary V_{DD} to the PLL (Phase Locked Loop), whose settling time is on the order of 10~100μs [20, 21]. Because the above characteristics result in large time granularity, these techniques are usually implemented for embedded processors that have lower requirements for performance. VSV, on the other hand, targets high-performance processors.

Recently, Magklis *et. al.* [19] propose a compiler-based voltage- and frequency-scaling technique, which targets multiple clock-domain processors. For each clock domain of the processor, the V_{DD} and the clock speed can

be changed separately. To determine the minimum V_{DD} and clock speed, the compiler inserts the reconfiguration instructions into applications by using profile-driven binary rewriting. In contrast, VSV is transparent to the software and does not need profiling. Xie *et. al.* [16] explores the opportunities and limits of compile-time dynamic voltage scaling by deriving an analytical model.

3. Circuit-level issues

In this section, we discuss the circuit-level issues related to varying V_{DD}.

3.1. Choice of supply voltages

Ishibara *et. al.* [4] show that the two-supply voltages is the most power-effective choice for variable supply-voltage designs. Therefore, we use the two-supply voltages. We refer to the two voltages as V_{DD}^{H} and V_{DD}^{L}.

We set V_{DD}^{H} to 1.8V, which is the normal supply voltage for the TSMC 0.18μm technology. V_{DD}^{L} should be chosen such that the corresponding clock speed is half of that at V_{DD}^{H}. This choice allows us to use a simple counter as a frequency divider, as opposed to varying the clock's PLL to divide the frequency by a factor that is not an integer. As mentioned before, changing the PLL is unacceptably slow [20, 21] for our context.

HSPICE simulation in TSMC 0.18μm technology shows that when V_{DD} is equal to 1.1V, the maximum clock speed is reduced to half of the clock speed corresponding to V_{DD}^{H}. To be conservative, we set V_{DD}^{L} to be 1.2V.

3.2. Rate of change of supply voltages

Burd *et. al.* [11] demonstrate that static CMOS is well-suited for continued operation during voltage transitions, and that dynamic logic styles can also be used as long as two conditions are satisfied: (a) V_{DD} ramps down by less than a diode drop V_{BE} during an evaluation phase of dynamic logic (eq.1); and (b) V_{DD} ramps up by less than V_{TP} (the threshold voltage of PMOS transistors) during the evaluation phase of dynamic logic (eq.2).

$$\frac{dV_{DD}}{dt} \geq \frac{-V_{BE}}{\tau_{clk}/2} \qquad (eq.1)$$

$$\frac{dV_{DD}}{dt} \leq \frac{V_{Tp}}{\tau_{clk}/2} \qquad (eq.2)$$

For the TSMC 0.18μm technology and a 1GHz clock, the dV_{DD}/dt limit is about 0.2V/ns. We use a more conservative dV_{DD}/dt rate of 0.05V/ns, for our experiments. Hence, switching between V_{DD}^{H}=1.8V and V_{DD}^{L}=1.2V costs 12 clock cycles.

3.3. Power-supply network

There are two options for the power-supply network: (1) using a single network and supply different V_{DD}'s at different times, or (2) using two different networks.

Present-day embedded systems that implement supply-voltage scaling use the first option. The supply voltage is continuously adjusted using a dynamic DC-DC converter. Unfortunately, the DC-DC converter requires a long time for voltage ramping [1, 14]. Since VSV targets high-performance processors running at GHz clock speeds,

the first option is not suitable for our purpose. Therefore, we choose the second option and use a dual-power-supply network to achieve fast V_{DD} switching. This structure includes two interwoven power-supply networks: one is used for V_{DD}^{H}, and the other for V_{DD}^{L}. Between each pair of connection points in a circuit module, a pair of PMOS transistors with complementary control signals is inserted to control voltage switching. Therefore, only one power-supply network is selected at one time for a particular module.

The additional supply lines in our dual-power-supply network introduce overhead in terms of energy and area. In Section 5.2, we calculate the energy overhead and show that it is minimal. Previous work [15] shows that *dual-V_{DD}* design using dual-supply lines causes about 5% area overhead.

3.4. Varying the clock speed

VSV raises three issues for the clock: (1) clock distribution, in the presence of varying clock speed; (2) clock power in the presence of varying V_{DD}; and (3) synchronization with off-chip circuitry. We consider each of these issues in turn.

First, as technology scales down, it is difficult to send a signal across the chip in a single cycle. Based on Matzke's estimation [12], the whole die is reachable within 2 clock cycles for 0.18μm technology. Therefore, we assume 4ns are needed for clock distribution – 2ns used to transmit the control signal to the root of clock tree, and 2ns for the clock to propagate through the clock tree.

Before a V_{DD}^{H} to V_{DD}^{L} transition, the clock speed is reduced first, even though the processor continues operating at V_{DD}^{H} for 4 more nanoseconds to distribute the slower clock. When ramping up from V_{DD}^{L} to V_{DD}^{H}, the sequence is reversed: the V_{DD} is ramped up first and then the faster clock is distributed. If this ordering is reversed, the processor will operate at full clock-speed but lower V_{DD}, causing functionality faults. In either case of ramp-up or ramp-down, the processor works at half clock-speed while the V_{DD} is changing. To slightly optimize performance, we overlap the distribution of the full-speed clock itself with the last 2ns of V_{DD} ramp-up.

Second, to avoid the PLL's slow settling time, we keep the PLL operating at V_{DD}^{H} to generate full-speed clock, which is sent to the root of the clock tree. When the processor switches to V_{DD}^{L}, we use the counter described in Section 3.1 to reduce the clock speed. To enable clock-power savings, the V_{DD} of the clock tree itself varies with the rest of the processor, although the PLL operates at V_{DD}^{H}. Lowering the V_{DD} of the clock tree does not hurt performance because at lower V_{DD} the clock speed itself is lower, leading to more slack in the clock period to accommodate the slower tree.

Finally, because the chip and the external devices already use different clocks and are naturally asynchronous, synchronizers exist at the interface even in a normal processor. Therefore, VSV will not introduce extra synchronization circuitry at the interface.

3.5. Varying the V_{DD} of RAM structures

Although supply-voltage scaling is effective for power reduction, we cannot implement it in all the sections of the processor. Ramping V_{DD} up and down involves charging and discharging of internal capacitances, respectively, which in turn introduces energy overhead. Supply-voltage scaling is feasible for a circuit block only when this energy overhead can be amortized by the power reduction achieved during V_{DD}^L operation.

For a RAM structure, during an access, only a small fraction of the cells are accessed, and only this fraction saves power when operating at V_{DD}^L. However, in a V_{DD} transition, *all* the cells either charge or discharge. Therefore, a large number of accesses at V_{DD}^L are needed to amortize the transition energy overhead.

The situation for combinational logic is different. Because the entire logic circuit is activated for each operation, the *entire* circuit (as opposed to the RAM's small fraction) achieves power saving while operating at V_{DD}^L. Consequently, even though a V_{DD} transition charges or discharges the entire circuit (like RAM), the power saving achieved at V_{DD}^L easily amortizes the transition overhead.

For example, for a 64KB 2-way L1 cache, all the SRAM cells are charged when V_{DD} ramps from V_{DD}^L to V_{DD}^H. The energy overhead of the transition is:

$$E_{overhead} = 64KBytes \cdot 8 \cdot C_i \cdot (V_{DD}^H - V_{DD}^L)^2 \cdot T_{CLK} \qquad (eq.3)$$

Here, C_i is the internal capacitance of a SRAM cell, and T_{CLK} is the cycle time of the clock.

In a two-way L1 cache access, 2 blocks are read – one in each way of the accessed set. If the block size is 32 bytes, then 2 x 32 x 8 bits are read. Hence, the energy savings during V_{DD}^L operation for a cache access (only dynamic power is considered) is:

$$E_{saving} = 2 \cdot 32 Bytes \cdot 8 \cdot C_i \cdot (V_{DD}^{H\,2} - V_{DD}^{L\,2}) \cdot T_{CLK} \qquad (eq.4)$$

$$\frac{E_{overhead}}{E_{saving}} = 1000 \cdot \frac{V_{DD}^H - V_{DD}^L}{V_{DD}^H + V_{DD}^L} \approx 200 \qquad (eq.5)$$

To compensate $E_{overhead}$, at least 200 accesses are needed at V_{DD}^L. Such number of accesses is highly unlikely because the processor would be stalled upon an L2 miss and only a few cache accesses would happen.

If we do a similar calculation for a combinational logic circuit with internal capacitance C_{logic}:

$$\frac{E_{overhead}}{E_{saving}} = \frac{C_{logic} \cdot (V_{DD}^H - V_{DD}^L)^2}{C_{logic} \cdot (V_{DD}^{H\,2} - V_{DD}^{L\,2})} = \frac{V_{DD}^H - V_{DD}^L}{V_{DD}^H + V_{DD}^L} \approx 0.2 \qquad (eq.6)$$

Hence, for combinational logic, the power saving of one operation at V_{DD}^L is much more than the energy overhead of a V_{DD} transition.

We see that supply-1voltage scaling should not be implemented for large RAM structures, such as the register file, I-cache, and D-cache. But for small RAM structures and combinational logic circuits, the energy overhead can be compensated.

3.6. Level conversion on the path from V_{DD}^L to V_{DD}^H

In the VSV processor, the register file and the L1 caches keep operating at V_{DD}^H, irrespective of whether the other sections of the pipeline are operating at V_{DD}^H or V_{DD}^L. Level-converting latches are needed when driving V_{DD}^H blocks with V_{DD}^L blocks (driving V_{DD}^L blocks with V_{DD}^H blocks does not need converters). Hence, level-converting latches are needed between the RAM structures and the pipeline.

Because a level-converting latch has a longer delay than a regular latch, simply replacing regular latches with level-converting latches would increase cache and register access time. Access time increases not only when driving V_{DD}^H RAM structures with V_{DD}^L blocks, but also when driving V_{DD}^H RAM structures with V_{DD}^H blocks, because signals still go through level-converting latches, even though no level conversion is needed.

Because the no-conversion path (i.e., when driving V_{DD}^H RAM structures with V_{DD}^H blocks) is used when operating at full clock-speed, the unnecessary delay due to the level-converting latches is a problem. To mitigate the delay, we place both regular and level-converting latches on the path between the structures and the blocks. Using a simple multiplexer, we choose the appropriate latch. Thus, the level-converting latches are bypassed when no conversion is necessary. To optimize the no-conversion path further, we implement the multiplexer by using a transmission gate that is placed at the output of the level-converting latch, leaving the regular latch latency virtually unchanged. Because transmission gates have only a small RC delay, even the level-conversion path does not incur much overhead due to the multiplexer.

However, the level-conversion path still incurs the overhead of level conversion and the transmission gate. We hide the latency by exploiting the following observation: Although the RAM structures are clocked at half clock-speed when the rest of the pipeline operates at V_{DD}^L, the structures still use V_{DD}^H. Consequently, their access times are still fast, creating a slack between the fast structures and the longer clock cycle. VSV uses this slack to absorb the level conversion latency.

Because the transmission gate selects only one set of either regular or level-converting latches at any time, the control signal for the transmission gates can be used to clock-gate the unselected latches. Hence, only one set of latches contributes to processor power.

4. Microarchitectural issues

Figure 1 depicts a processor that implements VSV. The processor is divided into two sections. The sections with large RAM structure (e.g. register file and I/D caches) and PLL are shown in gray. We do not implement voltage scaling in these areas because of either large power overhead (Section 3.5) or an unaffordably long settling time (Section 3.4). All the other sections shown in white apply VSV, and use a dual-power-supply network. The state machines (FSMs) monitor the issue rate to control switching of V_{DD}, as mentioned in the introduction.

The VSV processor has two steady operation modes: (a) In the *high-power mode*, the entire processor operates at V_{DD}^H and full clock-speed for the high-performance requirement; (2) and in the *low-power mode*, the processor runs at half clock-speed. To reduce power dissipation, most sections of the pipeline, except those shown in gray in Figure 1, operate at V_{DD}^L.
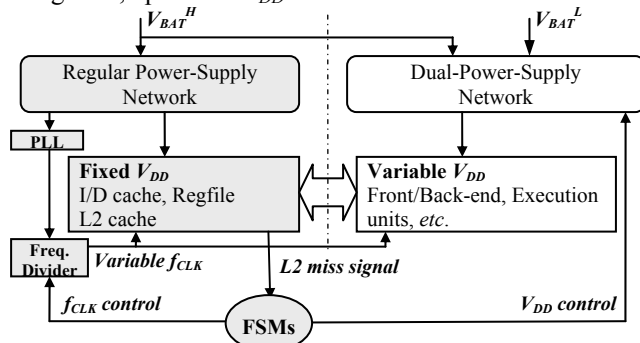

Figure 1 Structure diagram of VSV

4.1. High-power mode

In the high-power mode, the processor operates at V_{DD}^H and full clock-speed. This is the default mode of operation.

4.2. High-to-low power mode transition

VSV uses L2 cache misses to flag the absence of ILP. However, depending on the degree of ILP, the processor may find independent instructions to overlap with L2 misses. If the processor switches to the low-power mode in such high-ILP cases, significant performance degradation will ensue, as mentioned in the introduction. To avoid the performance loss, VSV employs *down-FSM* to monitor the instruction-issue rate, and works with the L2 miss signal, to guide switching to the low-power mode.

Down-FSM works as follows: when an L2 miss signal reaches the processor (after L2 hit latency), the FSM starts recording the instruction-issue rate for the small *period* (e.g., 10 cycles) that ensues. During this period, if the number of consecutive cycles in which no instruction is issued is above a *threshold* (e.g., 1-5 cycles), the transition to the low-power mode starts. The processor can begin transitioning as soon as the threshold is satisfied, without having to wait until the end of the monitoring period.

For maximum power savings and minimum performance loss, down-FSM should signal a high-to-low transition when and only when the long-term ILP after an L2 miss is low. The "when" part is needed to avoid a loss of power-saving opportunity, and the "only when" part is needed to avoid a loss of performance.

Two phenomena in modern processors – out-of-order issue and short-term parallelism of instructions near a missing load – can mislead down-FSM to violate the "when" part by mistakenly identifying ILP to be high when it is actually low. We observe that the absence of ILP during the monitoring period often signifies the absence of long-term ILP. Because the monitoring starts only *after* an L2 miss is detected, any independent

instruction due to this out-of-order issue and short-term parallelism has mostly already been issued from the time the missing load was issued to the time when the L2 miss was detected. Therefore, down-FSM is accurate in identifying the absence of ILP, despite the out-of-order issue and short-term parallelism.

Satisfying the "only when" part is straightforward because if the long-term ILP is high, then it often turns out that the ILP in the monitoring period is also high.

L2 cache misses caused by prefetches (software or hardware) do not cause long pipeline stalls. Therefore, VSV does not switch to the low-power mode during these misses.

Figure 2 illustrates the timeline of events in a high-to-low power mode transition, including the time taken for signal distribution (Section 3.4). The numbers in Figure 2 indicate latency in unit of nanosecond.
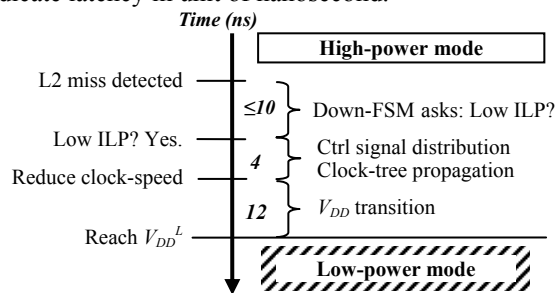

Figure 2 Timeline: High-to-low power mode transition

4.3. Low-power mode

In the low-power mode, the pipeline operates at V_{DD}^L and half clock-speed. Because caches and register file operate at V_{DD}^H even when the pipeline is in the low-power mode (Section 3.5), there is a choice of clocking these structures either at full or half clock-speed. Although clocking these structures at full clock-speed improves performance, this choice causes many difficulties.

This choice will result in two different clock speeds in operation at the same time – the pipeline operating at half clock-speed and the structures at full clock-speed. If this mismatch in clock speeds *always* were to exist, then accounting for the mismatch would amount to halving the structures' latencies, and there would be no major difficulties. However, when the pipeline goes back to operating at full clock-speed, these structures will continue to operate at full clock-speed, and the mismatch will disappear. Consequently, the latency of the structures counted in number of clock cycles of the pipeline varies with the power mode of the pipeline.

This variation raises four issues. First, modern pipelines exploit the deterministic cache hit (or register access) latency *predetermined* at design time for back-to-back scheduling of the instructions dependent on loads. Such scheduling is critical for high performance due to the frequent occurrence of loads. To exploit the full clock-speed of the cache, the scheduler would need to be modified to do back-to-back scheduling for two different

sets of latencies – one assuming X full-speed-clock cycles and the other assuming X/2 half-speed-clock cycles.

Second, the scheduler allocates resources, such as register write-ports, assuming a predetermined latency from the time when the instruction issues to the time when the instruction writes back. Such predetermined scheduling avoids extra-handshake overheads in common cases of load hits and stall-free execution. If the latency of the structures varies with the pipeline's power mode, the predetermined resource schedules would not work. For example, when the scheduler issues a load instruction, the resource scheduler reserves a register-write port for the cycle X, to write the returned value from the cache into the register file. However, because of the mismatch in speeds, the data cache returns the data earlier than the reserved time in the cycle X/2. The earlier arrival of the data would cause unexpected register-write port conflicts with the load that has scheduled the register-write port for that cycle. The scheduler would need to be changed to handle two different sets of schedules. Changes due to the first two issues may add complexity to the already intricate scheduler implementation.

Third, we assumed in Section 3.6 that the level-converter latencies can be hidden under the slack of the fast raw latency of the structures at V_{DD}^H and the half-clock-speed cycle. This assumption is violated if the structures are clocked at full speed.

Finally, to handle the situation in which the pipeline is operating at the half clock-speed and the structures are operating at the full clock-speed, we may need extra hand-shaking between the pipeline and the structures, which typically causes extra delay [21]. Because the hand-shaking circuitry is physically inserted on the cache hit critical path, designs which run in different clock-speeds may result in longer cache hit latency, even when the cache and the pipeline are running at the same speed in the high power-mode.

Therefore, we do not pursue the choice of using two different clock speeds. Instead, we let register file and L1 caches operate at the half clock-speed even though they are operating at V_{DD}^H (Section 3.1). By clocking the L1 cache and the register file at the same half clock-speed as the pipeline, L1 cache hit latency and register access latency will remain the same in terms of number of clock cycles of the pipeline, irrespective of the power mode. Thus, VSV avoids scheduler changes and other complications by sacrificing the opportunity to have shorter cache hit latency in the low-power mode.

Unlike the L1 cache, the hit latency of which is closely tied to the pipeline, the L2 cache is more loosely tied to the pipeline. In the case of the L2 cache, because the latency is longer and accesses are infrequent, there is no need for such a tight interface. Consequently, there is an asynchronous interface between the L2 cache and the pipeline. Recall that L2 cache always operates at V_{DD}^H, so its raw latency does not change. The L2 latency is X cycles when the processor is in the high-power mode and $X/2$

cycles when the processor is in the low-power mode. This change in cycle latency for the response from the L2 cache does not impact correctness and is automatically taken care of because of the asynchronous interface between the L2 cache and the processor.

4.4. Low-to-high power mode transition

When in the low-power mode, VSV uses the L2 cache miss returns to guide switching back to the high-power mode. If there is only one outstanding L2 miss, VSV switches back to the high-power mode when the miss is returned. However, the decision of switching back is more complicated when there are multiple outstanding misses.

A returned L2 miss causes its dependent instructions to become ready for issue. On one hand, if the processor switches back to the high-power mode when there are only a few ready instructions, the processor will soon stall again due to other outstanding L2 misses, and waste the opportunity to save more power. On the other hand, if the processor does not switch back to the high-power mode when there are many ready instructions, the performance of the processor will be degraded.

To identify both of these cases, VSV employs another FSM, called *up-FSM*, which monitors the instruction-issue rate, and works with the returning L2 miss, to guide switching back to the high-power mode.

Up-FSM works as follows: when an L2 miss data returns to the processor, the FSM starts to record the instruction-issue rate for the small period (e.g., 10 half-clock-speed cycles) that ensues. During this period, if the number of cycles in which at least one instruction is issued is above a threshold (e.g., 1-5 half-clock-speed cycles), the transition to the high-power mode starts. Similar to the high-to-low transition, the processor can begin transitioning to the high-power mode as soon as the threshold is satisfied, without having to wait until the end of the monitoring period.

For maximum power saving and minimum performance loss, up-FSM should signal a low-to-high transition when and only when ILP is high after an L2 miss returns. The "when" part is needed to avoid a loss of performance, and the "only when" part is needed to avoid a loss of power saving opportunity. In fact, both "when" and "only when" are straight-forward for the low-to-high transition, and up-FSM does not often get confused.
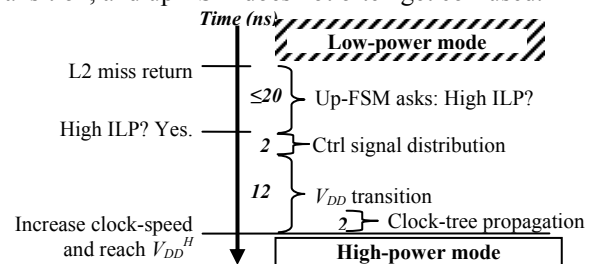

Figure 3 Time line: Low-to-high power mode transition

Figure 3 illustrates the timeline of events in a low-to-high transition, including the time taken for signal distribution (Section 3.4). The numbers in Figure 3

indicate latency in units of nanoseconds. Because the processor operates in half clock-speed when in the low-power mode, the maximum monitoring period is 20ns for the low-to-high transition.

5. Experimental Methodology

We used a modified version of Wattch [7] to simulate an 8-issue, out-of-order processor for evaluating VSV. To simulate accurately the timing effects of cache misses on VSV, we incorporated a memory bus model and MSHR (Miss Status History Register) into the simulator. To avoid underestimating the effect of the time to detect an L2 miss, we conservatively assume that the latency to detect an L2 miss is as long as the L2 cache hit latency. We warm up caches during fast-forward period, so VSV will not benefit from cold misses that do not occur in the stationary phase of program executions. Because most modern processors today use clock-gating [5] and software prefetching, we include these two techniques in all simulations.

Table 1: Baseline processor configuration

Processor	8-way issue, 128 RUU, 64 LSQ, 8 integer ALUs, 2 integer mul/div units, 4 FP ALUs, 4 FP mul/div units, uses clock gating (DCG) and s/w prefetching
Branch prediction	8K/8K/8K hybrid predictor; 32-entry RAS, 8192-entry 4-way BTB, 8 cycle misprediction penalty
Caches	64KB 2-way 2-cycle I/D L1, 2MB 8-way 12-cycle L2, both LRU
MSHR	IL1 - 32, DL1 – 32, L2 - 64
Memory	Infinite capacity, 100 cycle latency
Memory bus	32-byte wide, pipelined, split transaction, 4-cycle occupancy

We used pre-compiled, Alpha-SPEC2K binaries in [8], which are compiled with SPEC *peak* setting, including software prefetching. We used *ref* inputs in all simulations. We fast-forwarded two billion instructions, and then simulated one billion instructions. Table 1 lists the baseline configurations for all simulations.

5.1. Simulation of Hardware Prefetching Technique

Hardware prefetching techniques can effectively reduce cache misses to improve performance, on top of software prefetching. Because VSV exploits cache misses, we wanted to verify if VSV can still achieve power reduction in the presence of hardware prefetching.

In Section 6.4, we implemented the state-of-the-art, hardware-based Time-Keeping prefetching [9] in Wattch. Time-Keeping prefetching works as follows: If an L1 cache block is predicted dead, the predictor issues a prefetch request to lower memory hierarchy. The returned data block is placed in both L2 cache and a prefetch buffer close to L1, as described in [9]. The prefetch buffer is a 128-entry, fully associative cache that uses FIFO replacement policy, and has two-cycle access latency. The decay counters have a 16-cycle resolution. The 16KB address predictor uses nine bits from L1 tags and one bit from L1 indexes, to form signatures into the predictor. Because our baseline processor has a set-associative L1 data cache, we used per-set history traces for the address predictor, as recommended in [9].

5.2. Power Calculation

We estimated the overall processor power for 0.18μm technology by modifying the Wattch power equations to capture the effect of variable V_{DD}. All our results show only dynamic power. Leakage power is not included in our simulations, because leakage power is small for 0.18μm technology.

We monitored the dynamic change in V_{DD} and frequency in each section of the processor. Depending on the V_{DD} at a particular cycle, the simulator adds the correct value of dissipated power to the system power. During the ramp-up/down of supply, for each of the cycles, we took the average value of the supply voltage at the beginning and end of the cycle as the effective supply voltage contributing to processor power. As described in section 3.2, we assumed that the ramping period is 12 cycles.

We modeled our dual-power-supply network as a distributed *RLC* network by replacing each segment of the power grids with a lumped resistance-inductance-capacitance (*RLC*) element [13]. By using HSPICE, we simulated a dual-power-supply network for Alpha 21264 processor. To be consistent with 0.18μm technology, we shrank the layout of Alpha 21264 processor [6]. Our experiments showed that the dual-power-supply network structure can switch between V_{DD}^{H}=1.8V and V_{DD}^{L}=1.2V in 12ns (12 cycles f_{CLK}=1GHz) with 66nJ energy dissipation during ramping. We accounted for this energy overhead in our calculations.

For the latches on the path from V_{DD}^{L} to V_{DD}^{H}, the simulator adds the power of the regular latches in high-power mode, or the power of the level-converting latches in low-power mode (Section 3.6).

Because clock gating is widely used in commercial processor [5], we built baseline processor on deterministic clock gating [10], which clock gates the functional units, pipeline latches, D-cache wordline decoders and result bus drivers. Hence, for each of these circuits, if it is clock-gated in a particular cycle, it does not contribute to the overall power. On the other hand, if the circuit is not clock-gated, the simulator adds its dynamic power corresponding to the variable V_{DD}.

For the processor with timekeeping-based prefetching technique, we also included the power consumption caused by prefetching buffer in total power calculation.

5.3. L2 miss-rate

Table 2 shows instructions per full-speed-clock cycle (*IPC*) of SPEC2K benchmarks for the baseline processor, and L2 miss-rate, measured in number of L2 demand misses per 1,000 instructions (we call this number *MR*), for both baseline and Time-Keeping prefetching.

6. Results

We show the Wattch simulation results in this section. First, we present the effectiveness of VSV on the basis of power savings and performance degradations in Section 6.1. Next, we describe the effects of monitoring thresholds for high-to-low transition in Section 6.2 and for low-to-high

Table 2: Baseline SPEC2K benchmark statistics

	IPC	MR			IPC	MR	
	Base	Base	TK		Base	Base	TK
ammp	0.59	11.0	0.5	gzip	2.31	0.1	0.1
applu	2.32	10.1	4.1	lucas	1.34	10.2	4.2
apsi	2.51	1.4	0.7	mcf	0.29	67.4	48.2
art	1.36	10.3	11.7	mesa	3.64	0.3	0.2
bzip2	2.38	0.5	0.4	mgrid	4.17	1.5	0.8
crafty	2.68	0.0	0.0	parser	1.68	0.6	0.7
eon	3.13	0.0	0.0	perlbmk	1.41	1.3	0.6
equake	4.51	0.0	0.0	sixtrack	3.64	0.0	0.0
facerec	3.02	4.7	2.3	swim	3.81	5.8	1.4
fma3d	4.35	0.0	0.0	twolf	1.42	0.0	0.0
galgel	2.21	0.0	0.0	vortex	2.31	0.2	0.2
gap	3.00	0.5	0.3	vpr	1.25	2.0	2.1
gcc	2.27	0.1	0.1	wupwise	4.58	0.5	0.4

transition in Section 6.3. Finally, we show VSV's impact in the presence of Time-Keeping prefetching in section 6.4.

6.1. Effectiveness of VSV

In Figure 4 (top), we plot VSV's performance degradation for the SPEC2K benchmarks. The Y-axis corresponds to VSV's performance loss computed as a percentage of the execution time in the base processor running at full clock-speed. In Figure 4 (bottom), we plot the total CPU power savings (including the caches). The Y-axis represents power savings computed as a percentage of total power of the base processors. In both graphs, the X-axis shows benchmarks sorted by decreasing MR (L2 misses per 1,000 instructions). All runs include deterministic clock gating (DCG) and software prefetching. In the figures white bars show VSV without the FSMs and black bars show VSV with the FSMs. When operating without the FSMs, the processor goes to low-power mode whenever an L2 cache miss is detected, and goes to high-power mode whenever an L2 cache miss returns.
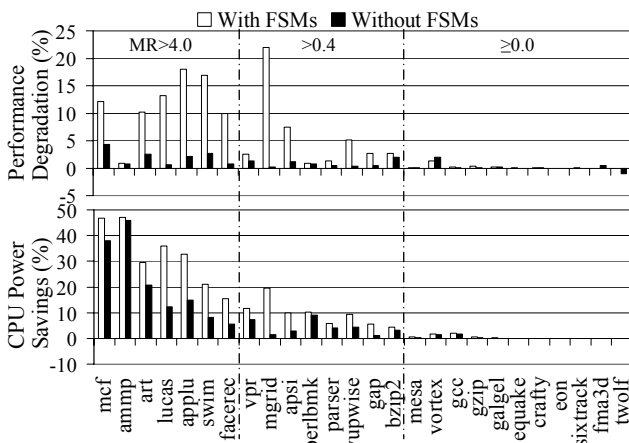

Figure 4 VSV Results: with and without the FSMs

We make three observations. First, VSV is effective in saving power, without (white bars) or with the FSMs (black bars). VSV saves average power by 33% without the FSMs and by 21% with the FSMs, for the benchmarks that have high MR. These benchmarks are in the left section of the graphs. In *ammp*, *mcf*, *art* and *applu*, VSV with FSMs saves 46%, 38%, 21% and 15% powers, with only 1%, 4%, 3% and 2% performance degradations,

respectively. Averaging across all SPEC2K benchmarks, VSV achieves 7% power saving with 1% performance degradation.

Second, the FSMs are effective in avoiding performance degradation, by not switching to low-power mode when ILP is high after an L2 miss. Some programs that have high ILP, such as *applu*, *swim* and *mgrid*, save 33%, 21% and 19% powers without the FSMs, albeit suffering performance degradations as high as 22%, 17% and 18%. With the FSMs, these programs give up much of their power savings, down to 15%, 8% and 1%, but now with small degradations of 2%, 3% and 0%, respectively. On average, benchmarks that have MR higher than 4 suffer 12% degradation without the FSMs, but only 2% with the FSMs.

VSV can still degrade performance despite using the FSMs. Performance degradations are caused by: 1) the down-FSM signals processor to enter low-power mode, despite high ILP after an L2 miss, 2) the monitoring period for low-to-high transition operates with reduced clock-speed, thus delaying critical path instructions. Despite these possible mishaps, all programs suffer less than 3% performance degradation, except *mcf* which suffers 4%. VSV effectively saves power for many programs with only 1% performance degradation.

Finally, for the benchmarks in the middle section of the graphs, VSV with FSMs achieves power savings between 1.1% and 8.9%, all at less than 2% performance degradations (e.g., *perlbmk* achieves 8.9% reduction in power with 0.8% performance degradation). VSV neither saves power nor degrades performance for programs that have low MR (right sections of the graphs), because they seldom operate in low-power mode.

Because our interest in VSV is for saving power with minimal performance degradation, we will describe the results only with the FSMs for the remainder of the paper.

6.2. Effects of thresholds for high-to-low transition

To see the effectiveness of issue-rate monitoring for high-to-low transitions of power modes, we varied the monitoring threshold as 0, 1, 3 and 5 cycle in Figure 5, and kept the monitoring period at 10 cycles. Recall that for high-to-low transitions a monitoring threshold of 3 means that down-FSM looks for 3 consecutive cycles where no instructions is issued in each cycle, and that the monitoring period of 10 cycles means that down-FSM looks for the threshold to occur within 10 cycles. Here, the cycles are in units of full-speed-clock cycle. For low-to-high transitions, we use up-FSM that has 3-half-speed-clock-cycle threshold and 10-half-speed-clock-cycle monitoring period in all runs, because they help down-FSM achieve best results. For clarity, we show only the subset of benchmarks whose MR is more than 4.

From Figure 5, we see that low thresholds give VSV more opportunity to save power; but also cause more performance degradation, and vice versa for high threshold.

The bar *Threshold 0* corresponds to a processor that does not use down-FSM for high-to-low transition. VSV

with a threshold of 0 causes performance degradation as high as 13% for *swim* and 10% for *applu*. When increasing the threshold from 1 to 3, *lucas*, *swim*, *applu* and *facerec* significantly reduce performance degradations, but also save much less power. *Mcf* and *art* significantly reduce performance degradations, but still achieve roughly the same power savings as threshold one. With a threshold of 5, all of these benchmarks incur similar performance degradations as with the threshold of 3, but with less power savings. To sum up, a threshold of 3 is the best-performing, because it incurs performance degradation less than 5%, but saves more powers than threshold 5.
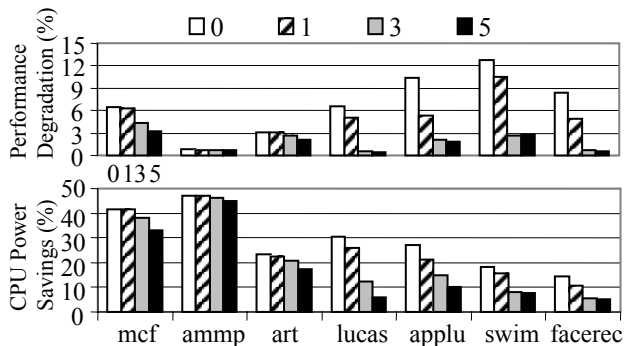
Figure 5 Effects of thresholds on high-to-low transitions

6.3. Effects of thresholds for low-to-high transition

To see the effectiveness of issue-rate monitoring for low-to-high transitions of power modes, we varied the corresponding threshold as 1, 3 and 5 cycles in Figure 6, and kept the corresponding monitoring period at 10 cycles. Recall that for low-to-high transitions a monitoring threshold of 3 means that up-FSM looks for 3 consecutive cycles where at least one instruction is issued in each cycle, and that the monitoring period of 10 cycles means that up-FSM looks for the threshold to occur within 10 cycles. Here, the cycles are in units of half-speed-clock cycle. For high-to-low transition, we keep the up-FSM at 3-cycle threshold and 10-cycle monitoring period in all runs. Again, we show only the subset of benchmarks whose MR is more than 4.

For comparison, we also show the results without monitoring, and instead use simple heuristics for making low-to-high transition whenever the first or the last outstanding L2 miss returns. To ease explanation, we call the former heuristic as *First-R* and the latter as *Last-R*. These simple heuristics trade-off performance and power: while Last-R saves more power than First-R, Last-R also causes more performance degradation than First-R.

From Figure 6, we see that increasing threshold value favors power saving while decreasing threshold favors performance. The results also show that the issue-rate monitoring is effective, because the monitoring achieves power saving close to Last-R's, and performance close to First-R's.

Threshold 1 incurs small performance degradation similar to First-R, but saves more power than First-R. A threshold of 3 achieves power savings closer to that of

Last-R, with performance degradation only 1~2% more than that of First-R, except in *swim*. A threshold of 5 behaves similar to Last-R both in performance and power, except in *swim* where the monitoring successfully reduces *swim's* performance degradation, from 13% to 5%.
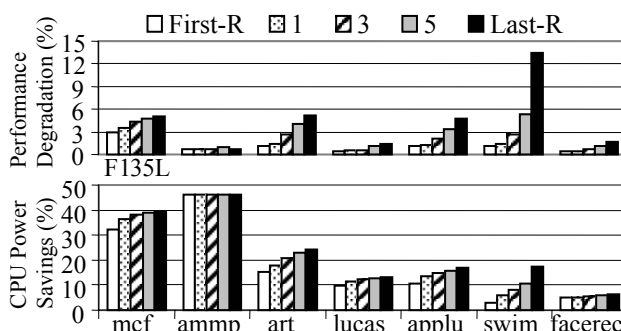
Figure 6 Effects of thresholds on low-to-high transitions

6.4. Impact of Time-Keeping prefetching technique

Time-Keeping Prefetching is an important, state-of-the-art technique to reduce cache misses. Although the technique is not a standard feature on modern processors, we stress-test VSV with the technique in this section, because the technique may remove cache misses to the point where VSV has no opportunity.
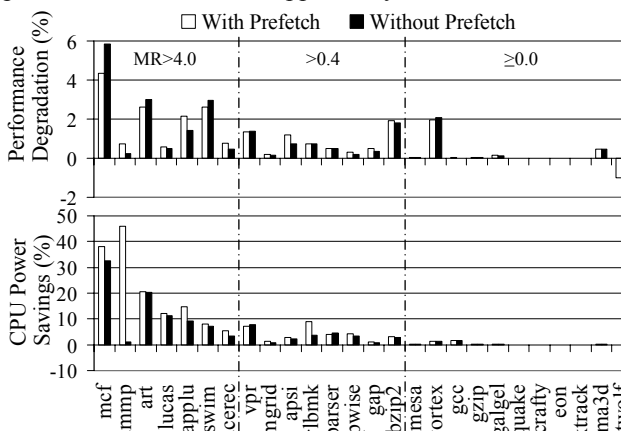
Figure 7 Impact of Time-Keeping Prefetching on VSV

Our experiment shows that although Time-Keeping prefetching reduces L2 miss-rates for most benchmarks, it does not remove all L2 misses. VSV could still save power for a processor that implements aggressive prefetching. On average, the prefetching reduces MR from 4.9 to 3.0.

In Figure 7, we show VSV's performance degradations and power savings without (white bar) and with (black bar) Time-Keeping, respectively. VSV uses the same FSM configuration as in Section 6.1.

Most benchmarks' performance degradations and power savings remain largely unchanged when prefetching is employed, except that *mcf's* degradation increases from 4.4% to 5.9%, and that *ammp's* power saving significantly reduces from 46% to 1%. *Ammp's* savings reduce because prefetching removes 95% of its L2 misses. Most of high-MR benchmarks roughly maintain their power savings,

because the prefetching does not significantly reduce the miss-rates for these benchmarks.

In summary, VSV achieves 12.1% average power saving with Time-Keeping, compared to 20.7% without Time-Keeping, for the benchmarks that have MR higher than 4. The average performance degradations remain at 2.1% with and without prefetching. Averaging across all SPEC2K benchmarks, VSV achieves 4.1% reduction in power with 0.9% performance loss. Hence, VSV can save power with minimal degradation, even for an aggressive processor that implements the latest, state-of-the-art prefetching technique.

7. Conclusions

We proposed a novel variable supply-voltage scaling (VSV) technique to reduce processor power without undue impact on performance. We proposed scaling down the supply voltage of certain sections of the processor during an L2 miss while being able to carry on the independent computations at a lower speed. We introduced two novel finite state machines (FSMs), *up-FSM* and *down-FSM*, that track parallelism on-the-fly. Down-FSM avoids performance degradation by preventing supply-voltage from scaling down when there are sufficient independent computations to overlap with the L2 miss. Up-FSM avoids reduction in power savings by preventing supply-voltage from scaling up when there are multiple outstanding misses and insufficient independent computations to overlap with them.

Circuit-level considerations impose many constraints on VSV. First, circuit-level complexity limits VSV to two supply voltages. Second, circuit stability and signal-propagation speed issues limit the rate of transition between the voltages and introduce a transition time overhead. Third, energy overhead factors disallow supply-voltage scaling of large RAM structures such as caches and register file upon an L2 miss.

For our simulations, we used the SPEC2K benchmarks executing on an eight-way, out-of-order-issue superscalar processor that implements clock gating and software prefetching. Our simulations showed that VSV with the FSMs reduces power by as much as 46% with performance degradation as small as 4% in the worst case. On average, VSV saves 7% in total processor power for all SPEC2K benchmarks, and 21% for benchmarks with high L2 miss rates (>4 misses per 1,000 instructions). We showed that the FSMs are effective in avoiding performance degradation for programs with high ILP. The FSMs reduce VSV's performance degradation from 12% to 2%, while limiting the drop in power savings from 33% to 21%.

Finally, we showed that VSV is effective in saving power, even in a processor that implements the aggressive Time-Keeping prefetching. VSV achieves 4% reduction in power for all SPEC2K benchmarks, and 12% for the benchmarks with high L2 miss rates, when both the baseline and VSV implement Time-Keeping.

8. References

[1] T. Burd, *et. al.*, "A dynamic voltage scaled microprocessor system", *IEEE Journal of solid-state circuits*, 35(11), pp.1571-1380.

[2] J. Pouwelse, *et. al.*, "Dynamic voltage scaling on a low-power microprocessor", *Mobile Computing Conference (MOBICOM)*, pp.251-259, Jul. 2001.

[3] C. Hsu, *et. al.*, "Compiler-directed dynamic frequency and voltage scheduling", *workshop on Power-Aware Computer Systems (PACS'00)*, pp.65-81, Cambridge, MA, Nov. 2000.

[4] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors", *Proc. of 1998 Int'l Symp. on Low Power Electronics and Design*, pp.197-202, Jul. 1998.

[5] M. Gowan, *et. al.*, "Power considerations in the design of the Alpha 21264 microprocessor", *Proc. of 35th Design Automation Conference (DAC)*, pp. 726 – 731, Jun. 1998.

[6] S. Manne, et al., "Alpha processors: A history of power issues and a look to the future", *Workshop of Cool Chip, in Conjunction with the 32nd In'l Symp. on Microarchitecture*, pp. 16-37, Nov. 1999.

[7] D. Brooks, *et. al.*, "Wattch: A framework for architectural-level power analysis and optimizations", *Proc. of 27th Int'l Symp. on Computer Architecture (ISCA)*, pp.83-94, 2000

[8] C. Weaver, SPEC2K Alpha binaries. http://www.eecs.umich.edu/~chriswea/benchmarks/spec2000.html

[9] Z. Hu, *et. al.*, "Timekeeping in the Memory System: Predicting and Optimizing Memory Behavior", *Proc. 29th In'l Symp. on Computer Architecture (ISCA)*, pp. 209-220, 2002

[10] H. Li, *et. al.*, "Deterministic clock gating for microprocessor power reduction", *In Proc. of 9th Int'l Symp. on High Performance Computer Architecture (HPCA)*, pp.113-122, Feb. 2003.

[11] T. Burd, and R. Brodersen, "Design issues for dynamic voltage scaling", *Proc. 2000 Int'l Symp. on Low Power Electronics and Design (ISLPED00)*, Jul. 2000, pp.9-14.

[12] D. Matzke. "Will physical scalability sabotage performance gains?", *IEEE Computer*, 30(9):37 – 39, Sep. 1997

[13] S. Zhao, K. Roy, and C-K. Koh, "Decoupling capacitance allocation and its application to power-supply noise-aware floorplanning", *IEEE Transation on Computer-Aided Design of Integrated Circuits and Systems*, 21(1), Jan. 2002, pp.81-92.

[14] A. Stratakos, "High-efficiency, low-voltage dc-dc conversion for portable applications", Ph.D. dissertation, UC-Berkely, CA, 1999.

[15] K.Usami, *et al.*, "Design Methodology of Ultra Low-power MPEG4 Codec Core Exploiting Voltage Scaling Techniques", *Proc. of 35th Design Automation Conference*, pages, pp. 483-488, Jun. 1998.

[16] Fen Xie, *et. al..* "Compile-Time Dynamic Voltage Scaling Settings: Opportunities and Limits", *Programming Language Design and Implementation (PLDI)*, June 2003.

[17] S. Tyagi *et al.*, "A 130 nm generation logic technology featuring 70 nm transistors, dual Vt transistors and 6 layers of Cu interconnects," *Digest of Technical Papers of International Electron Devices Meeting*, pp.567 -570, 2000.

[18] D. Marculescu, "On the use of microarchitecture-driven dynamic voltge scaling", *Proc. of the Workshop on Complexity-Effective Design,* Jun. 2000.

[19] G. Magklis, *et. al.*, "Profile-based dynamic voltage and frequency scaling for a multiple clock domain microprocessor", *Proc. of 30th Int'l Symp. on Computer Architecture (ISCA)*, pp.14-25, Jun. 2003.

[20] T. K. K. Kan, G. C. T. Leung, and H. C. Luong. "A 2-V 1.8-GHz Fully Integrated CMOS Dual-Loop Frequency Synthesizer", *IEEE Journal on Solid-State Circuits*, 37(8):1012 – 1020, Aug. 2002.

[21] G. Semeraro et, al, "Energy Efficient Processor Design Using Multiple Clock Domains with Dynamic Voltage and Frequency Scaling", *Proc. of 8th Int'l Symp. on High Performance Computer Architecture (HPCA)*, pp. 29-40, Feb. 2002.