

# Memory-Aware DVFS for CMP Systems

Saugata Ghose and Jonathan Tse

May 2009, Computer Systems Laboratory, Cornell University

{ghose, jon}@csl.cornell.edu

**Abstract**—High-performance processors are becoming increasingly power bound with technology scaling. Dynamic voltage and frequency scaling (DVFS) has emerged as an efficient method of reducing power consumption by lowering the operating voltage and frequency of a processor. We propose a multicore memory-aware DVFS scheme based on VSV, a uniprocessor DVFS algorithm that throttles a core based on L2 cache misses. The key observation is that during L2 misses, there may be periods during which the processor pipeline is stalled, waiting for data. These stalls offer an excellent opportunity for power savings with DVFS. Care must be taken, however, to be sure that the pipeline is actually stalled during an L2 miss and that the processor has sufficient work to complete when transitioning out of low-power mode. Using SPEC2K benchmarks, we evaluate both single-core and multicore VSV over a range of DVFS transition latencies: 12ns, 100ns, and 8.9 $\mu$ s which are representative of different voltage regulator configurations. We show that fast-switching, on-chip voltage regulators for DVFS are necessary to see benefits using the energy delay squared ( $ED^2$ ) metric. However, if low latencies—on the order of 12ns—are indeed possible, we show power benefits of 28%, performance costs of 35%, and  $ED^2$  improvements of 28% for a quad-core CMP. Increasing the latency to 100ns shows power savings of 45% at 45% performance loss, and additionally a  $ED^2$  degradation of 28%, though this is the result of poor prediction of instruction-level parallelism—8.9 $\mu$ s latencies proved to be entirely infeasible.

## I. INTRODUCTION

Over the last decade, the two major challenges of power consumption and memory limitations have drastically altered the landscape for processor design. Architects have long embraced the oft-quoted prediction of Gordon Moore, which dictates exponential growth for the number and density of transistors in an integrated circuit [1]. This exponential increase in transistor density—and design complexity—predicted by Moore’s Law, as it has since been dubbed, has essentially been directly correlated to increases in processor clock frequency. Until the turn of the century, this speed improvement directly translated to improved performance for end users. At that time, despite the continued growth of transistor counts on chip, processor frequencies began to level off. This decline is the direct consequence of two major barriers: the memory wall and the power wall.

### A. The Memory Wall

The memory wall was first recognized as a major issue by Wulf and McKee [2]. They observed that the growth in processor performance far outstripped growth seen in the performance of main memory. This disparity has only been exacerbated by the fact that despite conventional wisdom at the time [3], CMOS process technologies continue to shrink and provide raw processor performance improvements with which advances in memory design have not been able to keep pace [4].

Improving memory performance has been an area of ongoing research for decades. One of the early driving forces for this work was the significant growth of program scale and processor performance with time. One notable innovation from this period is the development of caches to drastically reduce latencies for temporal and spatial accesses to a subset of memory addresses [5]. Current research focuses on all aspects of the memory hierarchy, but there is still a significant focus on improving DRAM performance. Commodity products have also seen innovations such as double data rate (DDR) DRAM, which has gone through several iterations that attempt to maximize data bus utilization [6,7]. Significant research has also been directed towards memory access scheduling to improve performance. While a number of proposals have seen limited success [8,9,10], Rixner’s proposed scheduling algorithm [11] remains the preferred choice due to its competitive performance and simplicity.

In addition to performance-oriented improvements, a complimentary body of research has targeted addressing the impact that high-latency memory accesses have on the overall architecture. Work in this area involves latency-hiding and power-saving techniques such as load value speculation [12], leveraging DRAM sleep states [13], and more indirectly, improving instruction reordering [14] and throughput [15] within the processor. As detailed in Section II, our work builds on the premise that during a period when a pending memory access reduces the number of instructions that can be issued, the processor sits idle and unnecessarily consumes power.

### B. The Power Wall

Transistor density and processor clock speeds have reached the point where power dissipation is a major

problem and has become the focus of industry [16]. The power dissipation capabilities of die packaging have reached their limit, further compounded by the cubic relationship of power dissipation to clock frequency. As such, performance improvements can no longer feasibly be obtained by increasing processor frequencies [17]. This has resulted in a rethinking of the architecture domain, the most apparent result being industry’s adoption of the chip multiprocessor (CMP) as a solution to sustaining performance improvements [18]. CMPs are particularly attractive due to a higher potential for energy efficiency and a simpler design effort as compared to deeply-pipelined, highly-superscalar monolithic processors [19]. As a result, performance improvements are no longer free to software – programmers and operating systems are now responsible for extracting thread-level and task-level parallelism. Nevertheless, Amdahl’s Law dictates that in the ideal case, parallelized sections of a program can achieve speedups proportional to the number of cores being used.

Unfortunately, these ideal speedups are unattainable in practice. A major factor for this is the increased pressure that CMPs place on the shared memory hierarchy [20]. With an large number of executing threads and growth in software complexity, memory access contention increases drastically, with the problem worsening as additional cores are placed on die. The different nature of this problem degrades the effectiveness of traditional access latency mitigation techniques, and as a result, individual CMP cores will still have idle periods where memory accesses stall processor execution. This idle time will detrimentally affect power dissipation, effectively nullifying some of the energy benefits of using CMPs.

To address these issues, a major area of power-reduction research has been in the area of dynamic voltage and frequency scaling (DVFS) techniques. As power dissipation is a quadratic function of voltage and a linear function of frequency, controlling both voltage and frequency provides a mechanism to limit the amount of power drawn by a processor, albeit at the cost of performance. DVFS has seen success in single core systems [21,22] as well as CMP systems [23], but per-core DVFS for CMP incurs additional complexity and overheads [24]. A number of algorithms for automated DVFS have been proposed. Some of the most common DVFS schemes scale processor voltage during periods of inactivity or for latency-insensitive execution phases, which do not require full performance. One particular scheme, VSV [25], applies DVFS to a single-core processor during periods where pending L2 misses render the instruction pipeline idle. This technique, which forms the basis of our work, is described in more detail in Section II.

## II. RELATED WORK

A number of DVFS control schemes have been proposed. One common application domain is embedded processors, where power limitations have long been a driving factor [26]. One such case is a memory-aware DVFS scheme for embedded multimedia workloads [27], which is similar to VSV. DVFS has also been used to address thermal issues in processors. These dynamic thermal management (DTM) schemes [28,29] seek to introduce automated control mechanisms that reduce die temperature by throttling the processor while minimally impacting performance. Shye et al. propose using biometric feedback from end users in place of machine automated control to scale processor performance such that the user cannot perceive noticeable performance degradation [30]. As transistor sizes shrink, process variations have begun to have a more tangible impact on systems. With the recent advent of CMP architectures, per-core DVFS has been used to address the resulting power and performance differences [31,32]. There has also been some work done to tackle the issue of process variations on a finer grain, with the introduction of globally asynchronous, locally synchronous clock domains inside the processor core [33].

Li et al. propose variable supply voltage (VSV), which applies memory-aware DVFS to a single-core processor [25]. VSV uses L2 miss rates to determine processor activity, throttling down the processor if a L2 miss currently being serviced is stalling pipeline execution. However, due to out-of-order execution and instruction-level parallelism, naively scaling voltage and frequency on every L2 miss would hurt performance. If the instruction issue queue has enough instructions to continue execution, the slower speed would result in a significant penalty in throughput. To combat this, VSV uses a finite state machine (FSM) to monitor issue rates once an L2 miss is issued. The possibility exists that the issue rate may be high immediately after the L2 miss, but this could be transient, resulting in a significant drop in processor activity. Using an FSM counteracts this false-positive behavior by examining the trend in the rate. A similar FSM is used to determine if sufficient instruction-level parallelism (ILP) exists to justify ramping the processor back up to full performance once the L2 miss is serviced.

## III. PROPOSED SOLUTION

We propose to extend the VSV scheme described by Li et al [25] to work with CMP architectures. A characteristic of chip multiprocessors is that they share a low-level cache structure, typically the L2 cache. Though this sharing is compensated for by increasing cache size, CMPs still exhibit a higher amount of cache contention

[34]. More often than not, this contention translates into additional latencies for accesses to the shared structure, especially for multiprogrammed workloads, where the independent processes do not share memory locations. These increased access times will translate into larger windows of opportunity for multicore VSV to reduce processor power consumption.

#### A. CMP Implementation Challenges

Much of the fundamental VSV implementation can be ported from the single-core domain into CMPs. To allow per-core DVFS throttling, each processor has its own FSM to determine when each core should be throttled. In a CMP, the state machines are physically located within the individual cores, as the FSMs must count cycles relative to the frequency that the core is running at. By keeping the FSM within the core, the complexity of implementing DVFS will be lessened. These FSMs, described in more detail in Section IV-D, communicate with a consolidated DVFS controller, which then changes the voltage of the appropriate cores using on-chip per-core regulators.

The shared L2 cache in CMPs also offers a significant challenge for implementing multicore VSV. In a single-core processor, every L2 miss is generated by the same core, allowing the VSV FSMs to simply observe cache misses. In a CMP, this is no longer possible, as an L2 miss could be generated by any one of the cores. As multicore VSV uses per-core DVFS, these misses need to be associated with their requesting cores. This obstacle is overcome by using information in the miss status holding registers (MSHRs), which store a pointer to the originating L1 cache for each primary miss. Since L1 caches are associated with a particular core, the MSHRs allow the VSV controller to determine the originating core for the L2 cache miss.

Another issue involves the way in which latencies scale when a core is throttled down. As proposed in the single-core VSV implementation, the latencies for all of the integer and ALU function units are scaled with the processor frequency to take the same number of cycles to complete operations. However, single-core VSV proposes that cache latencies also be scaled down. In hardware, they effectively achieve this by maintaining cache voltages while reducing their operating frequency only. Such an implementation is not feasible in multicore VSV due to the shared L2 cache, which cannot be throttled down if only some of the cores have been scaled. As a result, multicore VSV does not reduce the frequency of the caches. Instead, as suggested in the VSV paper, we assume that the cache interface has additional logic that determines which of the two latencies to use for communications, introducing a nominal area overhead.

#### B. DVFS Transition Latency

There is a non-trivial latency cost incurred when switching DVFS states, which is the result of capacitance charge and discharge times, as well as PLL lock time. For the single-core VSV implementation, Li et al. claim that due to providing on-chip voltage regulators, using dual power supply rails, and using counters to halve frequency as opposed to having to re-lock the PLL, they can achieve 12ns DVFS transition latencies for their system. Additionally, the scheme proposed by Li et al. assumes the processor is still doing work during the transition process, as CMOS circuits can continue operation during voltage transitions [21]. In contrast, other DVFS implementations have indicated much greater switching delays. For more modern process technologies, off-chip voltage regulators require as much as  $8.9\mu\text{s}$  [24] to  $15\mu\text{s}$  [35] to achieve similar voltage scaling. Kim et al. propose an on-chip voltage regulator design that can achieve high-speed DVFS transition latencies closer to 100ns [23]. The costs of achieving such high-performance are the power and area overheads, which for a quad core system approach 17% and 3%, respectively.

Based on the works cited above, it seems that the latency assumed for the single-core VSV implementation was overly optimistic. Even with dual power supply rails, charge sharing and inrush current issues may overwhelm the on-chip regulators assumed by Li et al., which lack sufficient bulk capacitance in comparison to more robust off-chip regulators. Additionally, continuing processor operation during voltage transitions, while feasible, is made more difficult by their assumption that the scaled processor clock will be distributed simultaneously. These are important concerns, as isolated L2 misses will only throttle the core down for short periods. In such cases, a high overhead will result in excessive performance degradation. As a result, a range of latencies are examined for VSV, and the processor is assumed to stall during DVFS transitions, providing a more conservative basis to determine the viability of the VSV proposal. Specifically, we examine the following DVFS transition latencies: 12ns as used in single-core VSV, 100ns for the on-chip regulators proposed by Kim et al, and  $8.9\mu\text{s}$  for full off-chip regulation.

## IV. METHODOLOGY

#### A. Simulation Framework

The chosen simulation environment was a significantly modified version of the multicore SuperEScalar simulator (SESC) [36], which includes the Watch [37] and Cacti [38] power models as well as the HotSpot thermal model [39]. The simulator was originally modified to implement DTM [40], and as a result includes detailed power and temperature statistics. SESC was further

modified to include functional unit latency scaling for multicore VSV. One caveat to using this version of SESC is its usage of a simultaneous multithreading (SMT) processor model. In spite of restricting the simulator to perform only single-threaded execution, there are associated area and power overheads due to the existence of SMT structures. To accurately simulate operating conditions, two simulations were executed – the first runs were used to obtain steady state temperatures for the floorplans, which were then fed into the second runs as thermal starting points.

Table I  
BASELINE OUT-OF-ORDER PROCESSOR DETAILS

	Li et. al VSV	Single Core VSV	Multicore VSV
Process	180nm	70nm	
Max Frequency	1GHz	3GHz	
DVFS Frequency	500MHz	1.5GHz	
# of Cores	1		4
$V_{DD}$	1.8V	1V	
$V_{DD}^L$	1.2V	0.85V	
Pipeline Width	8-way	4-way	
Int/FP Reg.	N/A	80/80	160/160
Int/FP ALUs	8/4	4/2	
Int/FP Mult/Div	2/4	2/2	4/4
ROB Entries	128		
LSQ Size	64		
Branch Predictor	8K/8K/8K Hybrid	4K/4K/4K Hybrid	16K/16K/16K Hybrid
RAS Entries	32	64	
BTB	8192 entry, 4-way	2048 entry, 4-way	
Mispredict Penalty	8 cycles		
IL1 Cache	32KB 2-way, Private, LRU		
DL1 Cache	32KB 2-way, Private, LRU		
IL1/DL1 Latency	2 cycles		
L2 Cache	2MB 8-way, Private, LRU	4MB 8-way, Shared, LRU	
L2 Latency	12 cycles		
IL1/DL1 MSHRs	32/32	32/64	
L2 MSHRs	64		
DRAM Size	$\infty$	1GB	
DRAM Latency	100 cycles		
DVFS Latency	12ns	12ns/100ns/8.9 $\mu$ s	

### B. Processor Model

In order to realistically determine the performance of VSV, the architecture used in the single-core VSV paper was abandoned in favor of a modern processor specification. Two processor models were used—a 4-way issue, single core processor to evaluate and verify single-core VSV, and a 4-core, 4-way issue CMP. Table I lists the architectural details of both processor models. These 70nm processors include detailed floorplans that are used for power and thermal modeling.

Typical for this process technology, the processors are simulated with an operating frequency of 3GHz, which, like in single-core VSV, is reduced by half when DVFS

is enabled. Based on the IBM 10SF process—which at 65nm is very similar to our process—the processors were assumed to have a  $V_{DD}$  of 1.0V. SPICE simulations show appropriate values for  $V_t$  at 0.3V and the scaled voltage  $V_{DD}^L$  at 0.85V. As in the single-core VSV paper,  $V_{DD}^L$  is the minimum voltage required to execute at half the nominal frequency for the processor.

### C. Benchmarks

The SPEC CPU2000 (SPEC2K) integer and floating point benchmarks [41] were used to evaluate the VSV implementations. For each of the benchmarks, the simulator fast forwarded through the first 2 billion instructions in order to pass the initialization phase. The benchmarks were then executed for 500 million cycles at full frequency to evaluate VSV. It is important to note that the benchmarks are executed for 500 million cycles at 3GHz, regardless of current DVFS status. In other words, when VSV is active, the number of cycles apparent to the throttled processor will be less than 500 million, as the processor will be running at 1.5GHz for some time. Table II shows the behavior of the L2 cache for the benchmarks when simulated without VSV.

Table II  
BENCHMARK L2 MISS CHARACTERISTICS

	Per 1000 Instructions	Per 1000 Cycles	Per Access
<i>ammp</i>	64.4576	8.1081	0.9925
<i>applu</i>	5.8440	11.9820	0.4711
<i>apsi</i>	0.2056	0.6400	0.1332
<i>art</i>	2.3110	5.1089	0.0366
<i>bzip2</i>	0.8309	1.6567	0.0842
<i>crafty</i>	0.0226	0.0430	0.0010
<i>quake</i>	2.2481	3.9249	0.2971
<i>gcc</i>	0.1672	0.2444	0.0079
<i>mcf</i>	40.8150	18.5145	0.4433
<i>mesa</i>	0.5174	1.3061	0.0953
<i>mgrid</i>	2.0700	4.8130	0.2977
<i>parser</i>	0.6530	1.0515	0.0511
<i>swim</i>	11.2911	15.1755	0.2883
<i>twolf</i>	0.0308	0.0464	0.0012
<i>vortex</i>	0.2442	0.4884	0.0122
<i>vpr</i>	0.0196	0.0391	0.0013
<i>wupwise</i>	1.0403	1.8476	0.5003

#### Legend

Compute	Balanced	Memory
---------	----------	--------

For multicore VSV, six multiworkload benchmarks were created from the single-threaded SPEC applications. These application mixtures, selected to illustrate VSV behavior on both compute-intensive and memory-intensive workloads, are listed in Table III. The workload behaviors were determined by studying the miss ratio, which is the number of L2 misses per 1000 instructions—as seen in the first data column of Table II. Using the same guidelines as Li et al., a miss ratio greater than 4

indicates that a benchmark is memory-intensive, while a miss ratio less than 0.4 indicates that a benchmark is compute-intensive. Applications that fall in between the two bounds are considered to be balanced. As the experiments for multicore VSV were conducted according to cycle count as opposed to instruction count (to allow for accurate statistical analysis of the multiworkloads), the observed behavior differs somewhat from that presented by Li et al.

Table III  
MULTIWORKLOAD BENCHMARKS

Benchmark	Alias	Characteristics
<i>ammp-gcc-mesa-twolf</i>	<i>agmt</i>	M-C-B-C
<i>applu-parser-swim-vortex</i>	<i>apsv</i>	M-B-M-C
<i>apsi-art-quake-wupwise</i>	<i>aaew</i>	C-B-B-B
<i>bzip2-quake-mesa-mgrid</i>	<i>benm</i>	B-B-B-B
<i>swim-gcc-apsi-vortex</i>	<i>sgav</i>	M-C-C-C
<i>vpr-art-mcf-wupwise</i>	<i>vamw</i>	C-B-M-B

*B* : Balanced – *C* : Compute Bound – *M* : Memory Bound

#### D. Implementing Multicore VSV

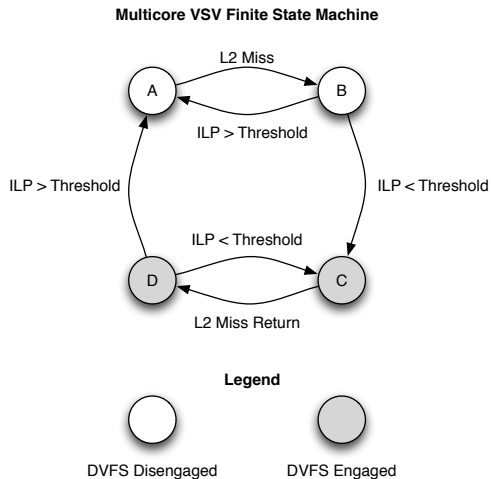


Figure 1. DVFS controller implemented as an FSM that monitors L2 miss rates and ILP—through instruction issue rates—as a measure of core activity. If ILP falls below a threshold, DVFS can be safely engaged, whereas high ILP indicates that the core has work to do despite the pending miss, so DVFS should not be engaged.

Data on L2 miss rates, L2 miss returns, and instruction issue rates per core are collected by instrumenting the L2 cache and issue window. This information is sent to the FSM inside each core and is updated each clock cycle. This approach is similar to the way Li et al. implemented single-core VSV, although we use a single FSM as opposed to a pair of complimentary FSMs as they did. Based on the available data, the FSMs determine which state to transition to, and then depending on the state, emit a decision on whether or not DVFS should be engaged for the core.

The FSM, as seen in Figure 1, has four states, labeled A through D. State A is the start state. If the core experiences an L2 miss, the FSM transitions to state B and begins monitoring instruction issue rates for a set window of cycles, which resets if a subsequent miss occurs. The FSM counts the number of cycles since the last instruction was issued, and if this crosses a threshold—3 cycles—the FSM transitions to state C, indicating that the processor is experiencing low ILP, and engages DVFS. If the number of cycles since the last issue remains low for the entire observation window of 10 cycles, ILP is seen to be sufficiently high and the FSM transitions back to state A, awaiting the next L2 miss.

The FSM remains in state C until a pending L2 miss is resolved, once data returns from main memory. When this happens, the state machine immediately transitions to state D and once again begins observing ILP by counting the number of cycles since the last instruction issue. If the ILP remains high for an observation window of 10 cycles, it is safe to disengage DVFS and transition to state A. However, if the number of cycles since the last issue exceeds 3 cycles, the FSM returns to state C to await the next L2 miss return.

## V. RESULTS

### A. Validating Single-Core VSV

As discussed in Section III-B, the issue of Li et al. using highly optimistic scaling latencies forces a reevaluation of the effectiveness of implementing VSV on a single-core processor. As previously mentioned, the processor model was updated to better match the parameters presented in other works for DVFS switching latency estimation [24,23] as shown in Table I. Initially, with the exception of the frequency and voltage, attempts were made to replicate all of the other parameters of the processor model used by Li et al. However, upon doing so, the simulation runs were found to exhibit thermal runaway due to floorplan incompatibility with the original VSV parameters. Though a number of parameters differ from the original VSV implementation in our final simulation framework, we believe that our models reflect a more realistic example of a modern processor.

Figure 2a shows the power dissipation for the single-core VSV configurations, normalized to baseline power dissipation for each of the benchmarks. Studying strictly power, the 12ns DVFS latency scheme shows an average power savings of over 32% across all of the benchmarks. The *apsi* benchmark shows a negligible decrease in power, indicating that VSV was not enabled often for the core. The best power savings are achieved by *twolf*, which reduces dissipation by 57%, despite the being compute-bound. Surprisingly, *twolf* saves more power

than the memory-bound benchmarks *ammp*, *applu*, *mcf*, and *swim*, which on average have a power savings of 29%.

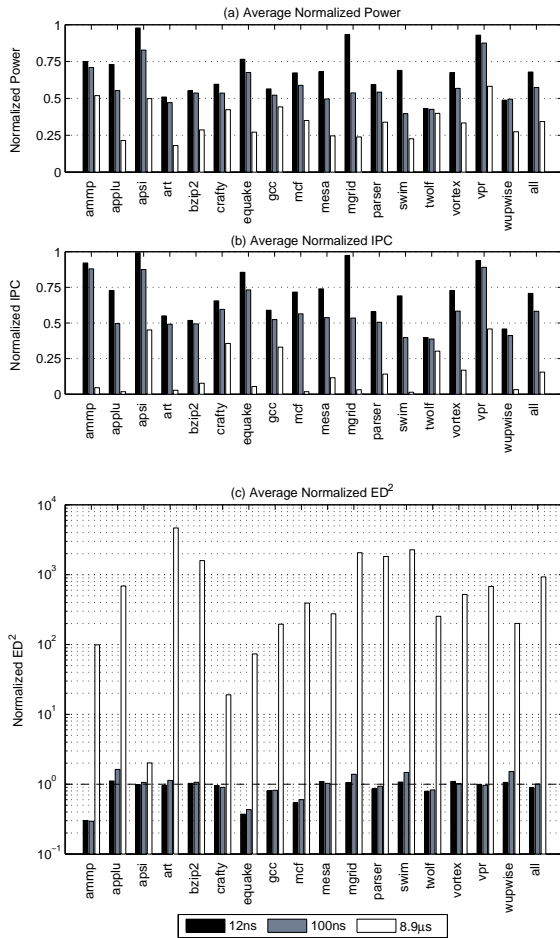


Figure 2. Average power (a), IPC (b), and  $ED^2$  (c) over the range of DVFS switching latencies for single-core VSV, normalized to a baseline core without DVFS.

Figure 2b, which illustrates the performance impact VSV has versus the baseline, provides better insight into why *twolf* sees such large power savings, especially in comparison to the memory-bound benchmarks. *twolf* experiences a performance degradation of over 60%, which corresponds to its power savings. This indicates that the benchmark is indeed compute-intensive, and that VSV is active when the processor could still be performing useful work. This is in contrast to the memory-bound benchmarks, which do not experience as severe a performance degradation. On the other hand, *apsi* experiences a 1% performance degradation. Figure 3 shows why this is the case—for *apsi*, the processor almost never throttles the core. The benchmarks which spend the most time in scaled mode are *twolf*, *ammp*, *wupwise*, and *mcf*, two of which are memory-bound. Figure 2b

shows that the memory-bound applications experienced moderate performance degradations, averaging to about 24%. Across the board, the average performance loss is over 29%. As seen in Figure 3, the benchmarks spend over 60% of their time in throttled mode.

Clearly, both performance and power are impacted significantly by VSV. For a more complete understanding, the energy-delay-square ( $ED^2$ ) metric can be used to quantify the costs at which the power savings are occurring. Figure 2c shows these values. This metric is especially valuable because of its ability to take into account the absolute differences. For example, if a benchmark saves significant power when it is throttled—at a high performance cost—but the baseline processor instructions per cycle (IPC) was low to begin with, the amount of energy saved will offset the slowdown, making the power reduction desirable. Such is the case with *twolf*—despite experiencing a 60% drop in IPC, enabling VSV results in an  $ED^2$  product that is 21% lower than the baseline. Overall, the average improvement across all benchmarks is 12%, with 10 of the 17 benchmarks showing on average a 37% improvement to their  $ED^2$  product. The worst benchmark is *applu*, which experiences an 11% increase in  $ED^2$ . Broken down by category, memory-intensive benchmarks experience a 25% reduction, compute-intensive benchmarks exhibit a 6% improvement, and balanced benchmarks improve  $ED^2$  by an average of 8%.

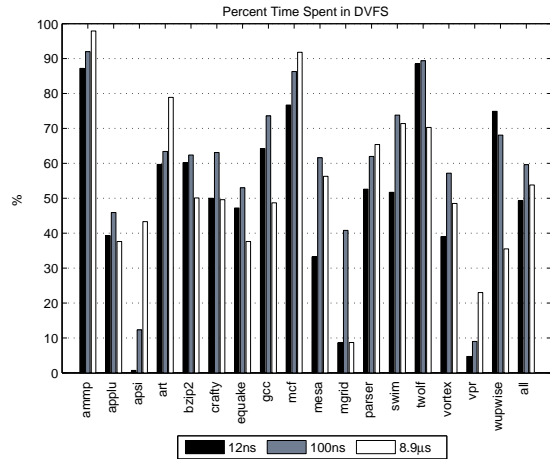


Figure 3. Percentage of time DVFS is activated by VSV.

### B. Evaluating DVFS Scaling Latencies

The results in Section V-A illustrate that without any changes to the FSMs, VSV can attain power savings as long as a 12ns DVFS transition is feasible. However, as discussed in Section III-B, such an assumption seems impractical. As a result, the single-core VSV implemen-

tation was also tested using 100ns and 8.9 $\mu$ s transition latencies.

As Figure 2a illustrates, when the switching latency is increased to 100ns, there is a notable increase in the power savings for all benchmarks. Much of this is the result of reducing dynamic power, as processor activity is suspended during the transitions, resulting in a minimization of transistor switching. With the change, *vpr* becomes the benchmark with the least power savings, at a 12% reduction, whereas *apsi* saves 17%. This is likely because of *apsi*'s significantly higher IPC, which would imply that in general, a greater amount of switching occurs inside the processor, and that stopping this switching would lead to greater power savings. This is corroborated by the sharper drop in normalized IPC for *apsi* over *vpr* in Figure 2b. As for *twolf*, its main power savings come from slowing down instruction execution, as opposed to reducing dynamic power dissipation due to the fact that *twolf* has a low L2 miss rate, as shown in Table II. Therefore, increasing the transitional latency will have little effect on *twolf*'s power and performance. On the other hand, *swim*, a memory-intensive application with a high L2 miss rate, sees its dynamic power decrease significantly, and as a result is now seeing the greatest power savings of all the benchmarks.

Performance takes an overall 12% hit for 100ns DVFS transition latency versus 12ns. This performance loss is directly related to the additional time that the processor sits idle as the voltage and frequency are transitioned. Unsurprisingly, *vpr* performs the best, as it spends the least time of all the benchmarks in throttled mode, as shown in Figure 3. Likewise, *twolf*, with its throughput crippled by DVFS, still exhibits the worst performance degradation at 61%. Studying the  $ED^2$  product shows the extent of the effects caused by the latency change. As seen in Figure 2c, *applu* and *wupwise* exhibit the greatest changes in the metric, increasing 47% and 43%, respectively. For *wupwise*, the change is the result of a 10% of the L2 misses occurring at the end of the 12ns run—as the simulator runs for a fixed number of cycles, these misses do not appear in the 100ns run. The number of L2 misses at the end of *applu* is even greater, with 33% of the misses occurring after the end of the 100ns execution, suggesting that *applu* experiences a phase change at that point. Had they been executed in the 100ns runs, the high concentration of misses could have been exploited by VSV to provide extensive power savings at a minimal performance cost. Overall, the  $ED^2$  product breaks even, only doing 0.5% worse on average. However, only 8 of the 17 benchmarks experience improvements.

Studying the results of VSV with 8.9 $\mu$ s DVFS transition latencies, it becomes immediately obvious that VSV cannot provide meaningful improvements with strictly

off-chip voltage regulators. As seen with the 100ns DVFS latency, power consumption continues to drop, this time being 66% less than the baseline. Only two benchmarks, *ammp* and *vortex*, have less than a 50% improvement. However, performance suffers an average 84% degradation. The extent of the impact can be seen in *parser*, which experiences an average drop in power, saving 66%. Across all of the physical units of the processor, dynamic power dissipation is reduced by a factor of 9 for the benchmark. This corresponds very closely to the drop in IPC for *parser*, which achieves only 14% of the performance that the baseline managed. The strongest argument for the invalidation of the off-chip regulator approach is the  $ED^2$  analysis in Figure 2c. All but one of the benchmarks experience at least an order of magnitude increase in  $ED^2$ —*art*, *bzip2*, *mgrid*, *parser*, and *swim* are three orders of magnitude higher. In fact, on average,  $ED^2$  increases by an astronomical factor of 930. The exception is *apsi*, a computation-bound benchmark, which experiences 50% power savings at the cost of only a 65% reduction in IPC, for a comparatively low  $ED^2$  increase of 101%. As previously mentioned, this is due to the fact that *apsi* transitions relatively infrequently in and out of scaled voltage and frequency mode. However, due to the extremely high overhead, the latency of these few transitions dominates, resulting in *apsi* spending about 60% of its execution time throttled, as Figure 3 illustrates.

Figure 3 also exhibits what initially appears to be an anomaly. For a number of benchmarks, the amount of time spent in low-power mode is less when 8.9 $\mu$ s transition latencies are used than when 100ns latencies are used. At first, this seems counter intuitive to the fact that additional overheads are being introduced to execution time. An extreme example of this is *mgrid*—for 100ns latencies, it spends nearly 60% of its time throttling the core, whereas only 17% of its time is spent in low-power mode at 8.9 $\mu$ s latencies. Further analysis of the execution reveals that when the latency is increased from 100ns to 8.9 $\mu$ s, the IPC drops 94%, i.e. 3% of the baseline. As a result, a much smaller window of instructions are executed by the simulator, which is likely unrepresentative of long-term program behavior. This impact directly affects the number of L2 misses issued per cycle, translating into fewer opportunities for VSV to enable DVFS. Comparing Figures 2b and 3 shows that all of the benchmarks that spend less time throttled for 8.9 $\mu$ s transition latencies exhibit a steep drop off in performance, and therefore the number of instructions executed.

### C. Multicore VSV

1) *Power Savings and Performance Costs*: The relative power savings for the six multiworkloads are shown

in Figure 4a. Unlike with the individual benchmarks, there is much less variation shown in the power savings, despite the differing balances of the workloads as described in Table III.

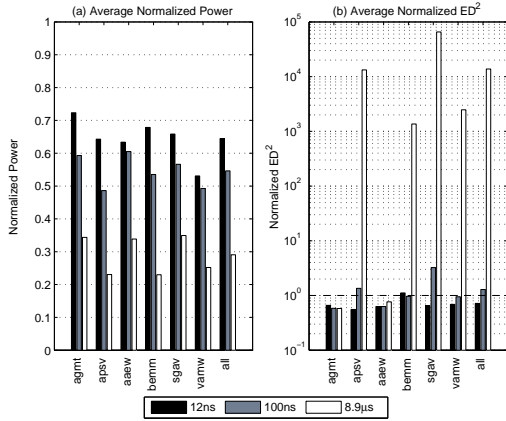


Figure 4. Average power (a) and  $ED^2$  (b) over the range of DVFS switching latencies for multicore VSV, normalized to a baseline core without DVFS.

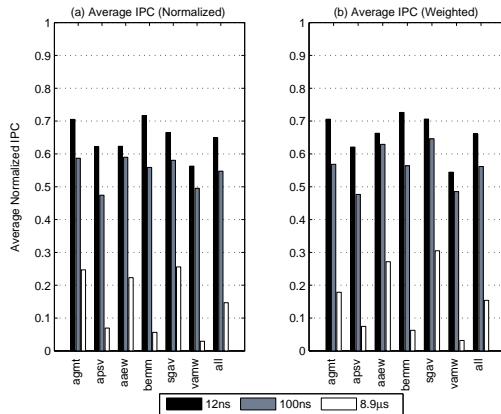


Figure 5. Average IPC calculated from the normalized values (a) and weighted for raw IPC (b) over the range of DVFS switching latencies for multicore VSV, normalized to a baseline core without DVFS.

For 12ns DVFS transition latencies, power savings of 35% are achieved over baseline, which is slightly better than the single-core results. Though the workloads are relatively evenly balanced between being memory- and computation-bound, of note is the fact that *apsv*, the most memory-intensive of the workloads, realizes the least power savings, at 28%. The performance degradation is shown in Figure 5, which shows both normalized performance treating each benchmark of the workload with equal importance—Figure 5a, and weighting the benchmarks in a workload by their absolute IPC—Figure 5b. For the 12ns latency, a 35% performance drop occurs. Interestingly, the normalized and weighted normalized

performance numbers are quite similar, indicating that no one benchmark dominates performance. Figure 4b illustrates the  $ED^2$  product for the workloads. As was the case with the single-core executions, the  $ED^2$  product averaged over all workloads performs better than the baseline by over 28% for the 12ns latency with one exception—the *bemm* workload, which spends the least time in DVFS mode as seen in Figure 6—with the improvement increasing to 36% on average excluding it.

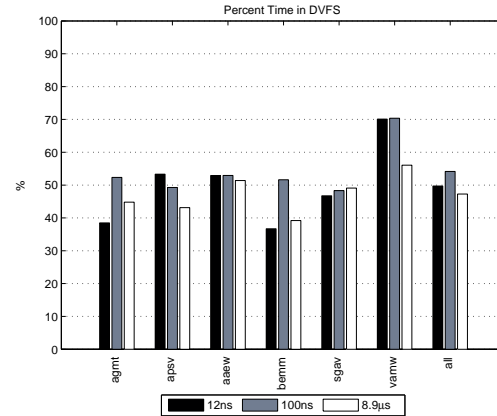


Figure 6. Percentage of time DVFS is activated by VSV for multiworkloads.

An analysis of the 100ns and 8.9 $\mu$ s transition latency results show similar trends to the single-core results. On average, power dissipation drops 45% and 70%, respectively, as shown in Figure 4a. Again, this power dissipation comes at the cost of significant drops in performance, which are more severe than for the single-core case. As shown in Figure 5a, the workloads see performance drops of 45% for 100ns transition latencies and 84% for 8.9 $\mu$ s latencies, with respect to baseline processor execution. Unfortunately, even at 100ns switching delays, VSV does not appear to be a viable option for these workloads, showing a 28% increase in the  $ED^2$  value over the baseline, according to Figure 4b. However, this average is skewed by the *sgav* workload, whose  $ED^2$  product increases by 223%. Much of this is the result of the *swim* benchmark, as Figure 7 shows that the power savings obtained came at a significant performance cost. Despite the classification of *swim* as memory-intensive, the benchmark still manages a relatively high throughput, with baseline IPCs of 1.195 in the multiworkload scenario. However, the benchmark still issues 6.5 L2 misses per 1000 cycles with 100ns DVFS switching latencies, and its IPC is likely not enough to guarantee that the FSM will never detect a low ILP situation. As a result, the benchmark experiences a high performance penalty when the L2 misses occur, resulting in the significant  $ED^2$  increase. Assuming this to be

a deficiency of the FSM—which is described in more detail in Section V-D—and calculating the average  $ED^2$  change for the other benchmarks, the workloads show a 10% improvement. As was the case with single-core VSV, the  $8.9\mu s$  latency  $ED^2$  product is an exorbitant 1378 times higher than the baseline. The *agmt* and *aaew* workloads still show improvements at these high latencies, however, which is attributed to their abilities to conserve significant dynamic power dissipation while still maintaining over 20% of the baseline performance.

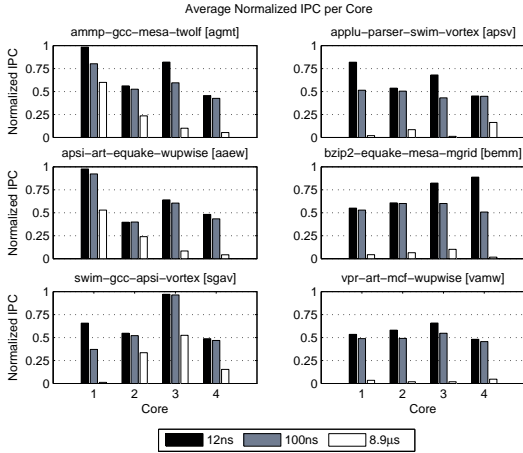


Figure 7. Average IPC per core over the range of DVFS switching latencies for multicore VSV, normalized to a baseline multiworkload without DVFS. Note that the multiprogrammed benchmark names are spatially mapped to which core ran which individual benchmark.

2) *L2 Cache Behavior*: As mentioned in Section III-A, L2 cache contention increases as more cores access the structure, which should therefore drive up both miss rates and latencies. Traditionally, this is counteracted by some combination of increasing the size or the associativity of the cache. For the processor model used for multicore VSV, as shown in Table I, the cache size was increased to 4MB, which is a typical value for modern quad-core processors. The *apsv* workload, with its more memory-intensive benchmarks is a good example. Breaking down the individual workloads, the memory-intensive benchmarks *applu* and *swim* experience modest L2 miss rate increases of 2-3%. However, *parser* has its miss rate increase from 5.1% to 23%, and *vortex* sees its miss rate jump from 1.2% to 33%. This indicates that *applu* and *swim* are still accessing memory more often than *parser* and *vortex*, and as a result, *parser* and *vortex* see their data evicted from the cache much more often. However, this does not translate into better opportunities for VSV—in fact, Figure 8 shows that with the exception of *vortex* for 12ns and 100ns transition latencies, the cores spend less time in low-power mode than they did for single-core VSV, shown in Figure 3. This is further corroborated by the

number of L2 misses that occur per 1000 cycles. For *applu* and *swim* combined in the 12ns latency case, 2 fewer L2 misses are issued per 1000 cycles, reducing the number of opportunities the benchmarks have to save power—the result of transition overhead. While an ideal instantaneous DVFS transition would be able to exploit the increased contention for added power savings, the performance overheads resultant from throttling the processor in effect nullify these extra opportunities.

In spite of these issues, multicore VSV was still able to outperform the savings achieved by the single-core implementation. From Figure 2c, the normalized average  $ED^2$  for the four benchmarks that make up *apsv* is 1.004 for a 12ns transition latency, yielding no improvement over the baseline. However, Figure 4b shows a significant reduction in the  $ED^2$  factor of 44%. One possible explanation is that with the added contention, the benchmarks stall more often—which results in the lower performance for the benchmarks as seen in Figure 7—and that the FSMs are better able to predict low ILP for this particular set of workloads. However, as the memory-intensive workloads see their performance drop further with longer transition latencies, multicore VSV no longer provides improvements for the  $ED^2$  product.

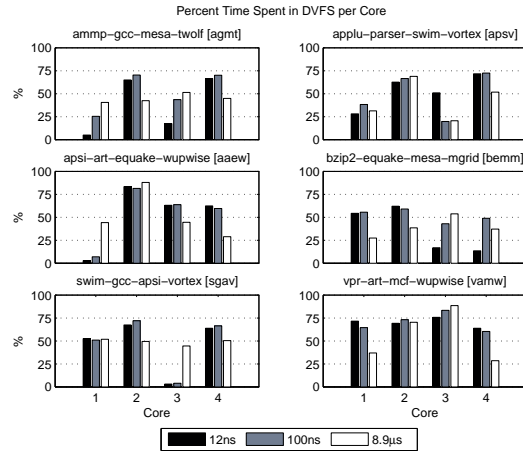


Figure 8. Percentage of time DVFS is activated per-core by VSV over the range of DVFS switching latencies for multicore VSV, normalized to a baseline multiworkload without DVFS. Note that the multiprogrammed benchmark names are spatially mapped to which core ran which individual benchmark.

There are some cases where multicore VSV does perform better than single-core VSV across the board. One such example is the *aaew* workload, where even for the  $8.9\mu s$  transition latency, the  $ED^2$  product improves over that of the baseline. Of interest is the fact that none of the benchmarks within the workload are memory-intensive. Studying the L2 cache behavior shows that overall, the cache experiences only a 4.1% miss rate. Though *apsi* maintains its low number of misses per 1000 cycles, the

number of misses per L2 access increases significantly, to 32%. Furthermore, as seen in Figure 7, *apsi* retains its high performance, which at 55% for an  $8.9\mu\text{s}$  switching latency is extremely high for a compute-intensive workload. This indicates that VSV is not scaling *apsi* to the point where it severely degrades its instruction throughput, even though, as Figure 8 shows, *apsi* is throttled for nearly 45% of the execution time. These statistics show that in this case, VSV is working effectively to reduce power consumption for these types of benchmarks. In fact, for the multicore workloads, the general trend is that workloads with more compute-intensive benchmarks show better improvements. Their higher IPCs prevent the FSM from mistakenly characterizing the workloads as having low ILP, yet they experience greater effects of cache contention because of the dominance of the cache by more memory-intensive benchmarks. As discussed in Section V-D, tuning the FSMs would likely allow VSV to provide better power savings for memory-intensive applications.

#### D. FSM Design Deficiencies

Figure 3 shows an interesting trend for the behavior of VSV. Even without high DVFS transition overheads, the 12ns latency single-core runs show that the benchmarks spent over 60% of their time in VSV mode. From Table II, the average number of L2 misses across all of the benchmarks is 4.41 misses per 1000 cycles. Assuming conservatively that all of these misses do not overlap, and that the latency for each is 112 cycles–100 cycle memory latency and one 12 cycle DVFS penalty as a rough average of the time the core is throttled, the time that DVFS should have been activated can be estimated to be at most 49% of execution time. Thus, the FSMs are currently engaging VSV mode for much longer than is necessary.

The situation gets worse when outliers are removed. Not considering benchmarks issuing more than 4 misses every 1000 cycles, VSV should only be active for 7.4% of the execution time, yet our results in Figure 3 show VSV is active for 57% of the time. This indicates that the FSM is doing a poor job of estimating when the processor needs to be throttled. Indeed, the benchmarks below the 4 miss per 1000 cycle threshold experience a 32% performance degradation. A likely reason for this is the fact that the parameters for the FSMs were designed for the highly superscalar processors that used by Li et al. As the issue widths for our cores are half that size, it is possible that the FSMs are incorrectly gauging periods of low ILP. It is not clear if this is strictly a result of poor parameter selection for the FSM, or if the general implementation of using such a simple FSM to predict ILP trends is flawed.

## VI. FUTURE WORK

There are a number of topics which are appropriate for further study. In particular, executing a set number of instructions as opposed to a set number of cycles would allow for a more fair basis of comparison between the various transition latencies. As described in Section V-B, some of the benchmarks exhibit changes in program phases, the behavior of which could be lost in a set-cycle simulation. Additionally, as discussed in Section V-A, VSV relies on the feasibility of fast-switching DVFS control, which, as discussed in Section III-B, seems infeasible. At the very least, an in-depth survey of current DVFS technology and transition characteristics warrants further study.

With the assumption that fast-switching DVFS control is possible, there are a number of parameter studies that are relevant. Due to the differences between our processor model and the one used by Li et al.—discussed in Section IV-B—the FSM issue counts and scheduled windows should be varied to re-tune its ILP prediction accuracy for our narrower superscalar processor, as mentioned in Section V-D. Additionally, varying the number of cores in the system to evaluate the effects of L2 contention for different multiworkloads could illustrate the scalability of VSV.

It should be noted that the main memory system for our implementation of SESC is extremely naive. The effects of memory latency on multicore VSV will be better gauged with the implementation of an accurate DDR2 model for main memory. Work could also be done on evaluating different memory access scheduling algorithms based on the knowledge of L2 behavior and instruction issue rates afforded by VSV, which could allow for the reordering of accesses to amortize DVFS transition penalties.

## VII. CONCLUSIONS

For the DVFS transition latency assumed by Li et al., VSV is able to provide significant power savings—32% across single-core benchmarks and 35% for multicore workloads—though these savings come with a significant cost in performance. This is in stark contrast of the Li et al. paper, which shows a 0.9% performance degradation. However, this is likely due to their use of an 8-way superscalar processor, which is optimistic for modern processors. Also optimistic is the 12ns DVFS transition latency assumed for their results. An evaluation of longer DVFS transition latencies shows that VSV loses much of its advantage when additional overheads are introduced. This is due to the fact that the limited duration of an L2 miss prevents high transition latencies from being amortized easily. Realistically, the latencies of off-chip voltage regulators are intolerable for this scheme,

severely degrading performance in both the single-core and multicore implementations of VSV. However, further analysis shows that the finite state machines implemented by VSV do a poor job of predicting ILP for the 4-way superscalar processors used in our evaluation. With improvements to their behavior, it is expected that using on-chip voltage regulators with 100ns switching delays may still yield power savings with a minimal impact to performance.

### VIII. ACKNOWLEDGEMENTS

The authors would like to acknowledge the contributions of Jonathan Winter, Derek Lockhart, Mark Cianchetti, and Robert Karmazin of Cornell University, and Engin Ipek of Microsoft Research. In particular, Jonathan Winter, Derek Lockhart, and Engin Ipek generously donated source code and notes to the project.

### REFERENCES

- [1] G. Moore, "Cramming more components onto integrated circuits," *Proceedings of the IEEE*, vol. 86, no. 1, pp. 82 – 85, Jan 1998.
- [2] W. Wulf and S. McKee, "Hitting the memory wall: implications of the obvious," *SIGARCH Computer Architecture News*, vol. 23, no. 1, Mar 1995.
- [3] M. Wilkes, "The memory wall and the CMOS end-point," *SIGARCH Computer Architecture News*, vol. 23, no. 4, Sep 1995.
- [4] S. McKee, "Reflections on the memory wall," *CF '04: Proceedings of the 1st conference on Computing frontiers*, Apr 2004.
- [5] J. Liptay, "Structural aspects of the system/360 model 8, part ii: The cache," *IBM Systems Journal*, vol. 7, no. 1, pp. 15–21, 1968.
- [6] "JESD79F JEDEC standard for DDR SDRAM," *JEDEC Solid State Technology Association*, 2008.
- [7] "JESD79-3C JEDEC standard for DDR3 SDRAM," *JEDEC Solid State Technology Association*.
- [8] O. Mutlu and T. Moscibroda, "Stall-time fair memory access scheduling for chip multiprocessors," *MICRO '07: Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec 2007.
- [9] —, "Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems," *ISCA '08: Proceedings of the 35th International Symposium on Computer Architecture*, Jun 2008.
- [10] J. Shao and B. Davis, "A burst scheduling access reordering mechanism," *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*, pp. 285–294, 2007.
- [11] S. Rixner, W. Dally, U. Kapasi, P. Mattson, and J. Owens, "Memory access scheduling," *ISCA '00: Proceedings of the 27th annual international symposium on Computer architecture*, Jun 2000.
- [12] G. Reinman and B. Calder, "Predictive techniques for aggressive load speculation," *MICRO 31: Proceedings of the 31st annual ACM/IEEE international symposium on Microarchitecture*, Nov 1998.
- [13] I. Hur and C. Lin, "A comprehensive approach to DRAM power management," *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, pp. 305–316, 2008.
- [14] J. Fisher, "Very long instruction word architectures and the ELI-512," *ISCA '83: Proceedings of the 10th annual international symposium on Computer architecture*, Jun 1983.
- [15] D. Tullsen, S. Eggers, and H. Levy, "Simultaneous multithreading: Maximizing on-chip parallelism," *Computer Architecture, 1995. Proceedings. 22nd Annual International Symposium on*, pp. 392 – 403, Dec 1994.
- [16] S. Naffziger, "High-performance processors in a power-limited world," *VLSI Circuits, 2006. Digest of Technical Papers. 2006 Symposium on*, pp. 93 – 97, Jan 2006.
- [17] D. Brooks, P. Bose, S. Schuster, H. Jacobson, P. Kudva, A. Buyuktosunoglu, J. Wellman, V. Zyuban, M. Gupta, and P. Cook, "Power-aware microarchitecture: design and modeling challenges for next-generation microprocessors," *Micro, IEEE*, vol. 20, no. 6, pp. 26 – 44, Nov 2000.
- [18] K. Olukotun, B. Nayfeh, L. Hammond, K. Wilson, and K. Chang, "The case for a single-chip multiprocessor," *ASPLOS-VII: Proceedings of the seventh international conference on Architectural support for programming languages and operating systems*, Dec 1996.
- [19] M. Oskin, "The revolution inside the box," *Communications of the ACM*, vol. 51, no. 7, p. 70, Jul 2008.
- [20] C. Liu, A. Sivasubramaniam, and M. Kandemir, "Organizing the last line of defense before hitting the memory wall for CMPs," *High Performance Computer Architecture, 2004. HPCA-10. Proceedings. 10th International Symposium on*, pp. 176 – 185, Jan 2004.
- [21] T. Burd, T. Pering, A. Stratakos, and R. Brodersen, "A dynamic voltage scaled microprocessor system," *Solid-State Circuits, IEEE Journal of*, vol. 35, no. 11, pp. 1571 – 1580, Nov 2000.
- [22] J. Pouwelse, K. Langendoen, and H. Sips, "Dynamic voltage scaling on a low-power microprocessor," *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, Jul 2001.
- [23] W. Kim, M. Gupta, G.-Y. Wei, and D. Brooks, "System level analysis of fast, per-core DVFS using on-chip switching regulators," *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, pp. 123 – 134, Jan 2008.
- [24] P. Juang, Q. Wu, L.-S. Peh, M. Martonosi, and D. Clark, "Coordinated, distributed, formal energy management of chip multiprocessors," *Low Power Electronics and Design, 2005. ISLPED '05. Proceedings of the 2005 International Symposium on*, pp. 127 – 130, Jul 2005.
- [25] H. Li, C.-Y. Cher, T. Vijaykumar, and K. Roy, "VSV: L2-miss-driven variable supply-voltage scaling for low power," *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, pp. 19 – 28, Jan 2003.
- [26] T. Martin and D. Siewiorek, "Nonideal battery and main memory effects on CPU speed-setting for low power," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 9, no. 1, pp. 29 – 34, Feb 2001.
- [27] J. Choi and H. Cha, "Memory-aware dynamic voltage scaling for multimedia applications," *Computers and Digital Techniques, IEE Proceedings -*, vol. 153, no. 2, pp. 130 – 136, Feb 2006.
- [28] M. Huang, J. Renau, S.-M. Yoo, and J. Torrellas, "A framework for dynamic energy efficiency and temperature management," *Microarchitecture, 2000. MICRO-33. Proceedings. 33rd Annual IEEE/ACM International Symposium on*, pp. 202 – 213, Nov 2000.
- [29] D. Brooks and M. Martonosi, "Dynamic thermal management for high-performance microprocessors," *High-Performance Computer Architecture, 2001. HPCA. The Seventh International Symposium on*, pp. 171 – 182, Dec 2000.
- [30] A. Shye, Y. Pan, B. Scholbrock, J. Miller, G. Memik, P. Dinda, and R. Dick, "Power to the people: Leveraging human physiological traits to control microprocessor frequency," *MICRO '08: Proceedings of the 2008 41st IEEE/ACM International Symposium on Microarchitecture*, vol. 00, Nov 2008.
- [31] R. Teodorescu and J. Torrellas, "Variation-aware application scheduling and power management for chip multiprocessors," *Computer Architecture, 2008. ISCA '08. 35th International Symposium on*, pp. 363 – 374, May 2008.

- [32] S. Herbert and D. Marculescu, "Variation-aware dynamic voltage/frequency scaling," *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, pp. 301 – 312, Jan 2009.
- [33] G. Semeraro, G. Magklis, R. Balasubramonian, D. Albonesi, S. Dwarkadas, and M. Scott, "Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling," *High-Performance Computer Architecture, 2002. Proceedings. Eighth International Symposium on*, pp. 29–40, 2002.
- [34] D. Chandra, F. Guo, S. Kim, and Y. Solihin, "Predicting inter-thread cache contention on a chip multi-processor architecture," *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*, pp. 340– 351, 2005.
- [35] R. Bergamaschi, G. Han, A. Buyuktosunoglu, H. Patel, I. Nair, G. Dittmann, G. Janssen, N. Dhanwada, Z. Hu, P. Bose, and J. Darringer, "Exploring power management in multi-core systems," *Design Automation Conference, 2008. ASPDAC 2008. Asia and South Pacific*, pp. 708 – 713, Feb 2008.
- [36] P. Ortego and P. Sack, "SESC: SuperESCalAr Simulator," *Reference Document*, Jan 2004.
- [37] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," *Computer Architecture, 2000. Proceedings of the 27th International Symposium on*, pp. 83 – 94, Jan 2000.
- [38] D. Tarjan, S. Thoziyoor, and N. Jouppi, "CACTI 4.0," *HP Laboratories*, Jan 2006.
- [39] K. Skadron, M. Stan, K. Sankaranarayanan, and W. Huang, "Temperature-aware microarchitecture: Modeling and implementation," *ACM Transactions on Architecture and Code Optimization (TACO . . .)*, Jan 2004.
- [40] J. Winter and D. Albonesi, "Addressing thermal nonuniformity in SMT workloads," *Transactions on Architecture and Code Optimization (TACO*, vol. 5, no. 1, May 2008.
- [41] J. Henning, "SPEC CPU2000: measuring CPU performance in the new millennium," *Computer*, vol. 33, no. 7, pp. 28 – 35, Jul 2000.