VRSync: Characterizing and Eliminating Synchronization-Induced Voltage Emergencies in Many-core Processors*

Timothy N. Miller, Renji Thomas, Xiang Pan, Radu Teodorescu
Department of Computer Science and Engineering, The Ohio State University
{millerti, thomasr, panxi, teodores}@cse.ohio-state.edu
http://arch.cse.ohio-state.edu

Abstract

Power consumption is a primary concern for microprocessor designers. Lowering the supply voltage of processors is one of the most effective techniques for improving their energy efficiency. Unfortunately, low-voltage operation faces multiple challenges going forward. One such challenge is increased sensitivity to voltage fluctuations, which can trigger so-called "voltage emergencies" that can lead to errors. These fluctuations are caused by abrupt changes in power demand, triggered by processor activity variation as a function of workload.

This paper examines the effects of voltage fluctuations on future many-core processors. With the increase in the number of cores in a chip, the effects of chip-wide activity fluctuation – such as that caused by global synchronization in multithreaded applications – overshadow the effects of core-level workload variability. Starting from this observation, we developed VRSync, a novel synchronization methodology that uses emergency-aware scheduling policies that reduce the slope of load fluctuations, eliminating emergencies. We show that VRSync is very effective at eliminating emergencies, allowing voltage guardbands to be significantly lowered, which reduces energy consumption by an average of 33%.

1. Introduction

Power consumption is one of the most significant roadblocks to future technology scaling according to projections from the International Technology Roadmap for Semiconductors (ITRS) [11]. Lowering supply voltage is one of the most effective techniques for improving energy efficiency, as evidenced by recent breakthroughs in low-voltage process technology in industry [9, 15]. Unfortunately, low-voltage operation faces multiple challenges going forward.

One of these challenges is increased sensitivity to voltage fluctuations. These fluctuations are caused by abrupt changes in power demand triggered by processor activity variation with workload. If the voltage deviates too much from its nominal value, it can lead to so-called "voltage emergencies," which can cause timing violations and memory retention errors in the processor. To prevent these emergencies, chip designers add voltage margins that in modern processors can be as high as 20% [12, 26], leading to higher power consumption than necessary.

Previous work [4, 5, 6, 7, 13, 24, 25, 26] has proposed several hardware and software mechanisms for reducing the slope of current changes (dI/dt), which dampens voltage fluctuations. This allows the use of smaller voltage guardbands, saving substantial amounts of power. All previous work, however, has focused on single-core [4, 6, 7, 13, 24, 25] or low core-count systems [4, 26].

In this work we show that, as the number of cores in future CMPs increases, the effects of chip-wide activity variation will overshadow the effects of within-core workload variability. The power demand of individual cores will account for a much smaller fraction of the chip's total power consumption. As a result, core-local activity is less likely to cause large power fluctuations that lead to emergencies. However, chip-wide coordinated activity such as that forced by global synchronization in multithreaded applications leads to much larger and rapid power fluctuations. For instance, barrier synchronization causes blocked threads to idle with very low power consumption. When all idle threads are released from a barrier the associated jump in activity across all cores leads to very large power spikes that can lead to voltage emergencies. Going forward, we will need to rethink the mechanisms used to avoid voltage emergencies to ensure they are effective and energy-efficient as the number of cores continues to scale.

This paper characterizes voltage variability in large CMPs running at low voltages. It shows that, in a 32-core CMP running multithreaded benchmarks, the most severe voltage droops are associated with thread synchronization primitives such as barriers and, to a lesser extent, with other thread activity such as new thread spawning. We find that about half of the benchmarks tested (a mix of SPLASH2 and PARSEC applications) trigger multiple emergencies in system with a typical 10% voltage guardband.

*This work was supported in part by the National Science Foundation under grant CCF-1117799 and an allocation of computing time from the Ohio Supercomputer Center.

Starting from this observation, we propose *VRSync*, a voltage-aware synchronization methodology that controls thread activity in critical scenarios. For example, *VRSync* enforces the gradual release of threads from barriers, leading to a gradual increase in power consumption. This limits the amplitude of the largest voltage droops, avoiding emergencies with smaller guardbands. *VRSync* is a software-only solution that can be implemented in system-level synchronization libraries and/or the OS. *VRSync* eliminates all emergencies in the benchmarks we test, allowing for a lower voltage guardband. A CMP with *VRSync* and a small guardband uses 33% less energy than a system that uses voltage guardbanding alone to eliminate emergencies. The runtime overhead of *VRSync* varies greatly with the density of synchronization and other application behavior, but averages about 6%.

Overall, this paper makes the following contributions:

- Analyzes the effects of thread synchronization on supply voltage stability. To the best of our knowledge, this is the first work to identify synchronization events as a major source of severe voltage droops in large CMPs. These observations are validated with power measurements from a 4-core, Intel Core i7 system.
- Shows that synchronization-induced emergencies are more likely as the number of cores increases.
- Presents *VRSync*, a novel synchronization methodology that prevents voltage emergencies, allowing smaller guardbands and saving energy.
- Evaluates *VRSync* using a commercial regulator model.

The rest of this paper is organized as follows: Section 2 provides some background on the design and limitations of voltage regulators. Section 3 analyzes synchronization-induced voltage droops in CMPs. Section 4 details the design and implementation of *VRSync*. An experimental evaluation is presented in Sections 5 and 6. Section 7 discusses related work, and Section 8 concludes.

2. Power Delivery and Regulation

Power to a modern CPU is delivered and controlled by a voltage regulator circuit. The regulator performs two main functions: It steps down the supply voltage to the level required by the CPU, and it keeps the voltage stable under varying current loads. Regulation is typically achieved by charging a capacitor on a duty cycle, using a low pass RLC filter to integrate the resulting voltage. The regulator monitors the output voltage and compares it to a reference voltage. When deviations occur (e.g. due to a change in load) it adjusts the duty cycle to maintain a stable output voltage.

2.1. Voltage Droops

Regulators are designed to respond quickly and precisely to changes in current loads, to prevent voltage fluctuations outside a narrow band around the nominal V_{dd} (typically $\pm$ 10%). Response time is, however, constrained by capacitor sizes, propagation latency (regulators generally reside off-chip) and by regulator switching frequencies. When load changes are small, the regulator easily controls the amplitude of the fluctuations. Figure 1(a) shows regulator response to an increase of $25\ \mathrm{Amps}/\mu\mathrm{s}$. Details on the regulator simulation are provided in Section 5. The supply voltage initially droops slightly, but the regulator quickly responds and prevents the output V_{dd} from decreasing by more than 10% of the nominal V_{dd}. In addition, some of the smaller high-frequency load fluctuations are generally absorbed by on-chip decoupling capacitors.

By contrast, Figure 1(b) shows regulator response to a larger load change ($45\ \mathrm{Amps}/\mu\mathrm{s}$). In this case, the magnitude and rate of increase are too great for the regulator, and we observe a droop that exceeds the safety margin and leads to an emergency. In our model, this current increase corresponds to about eight cores going from power-off to max power in one microsecond. In reality individual cores would not normally see such a large and abrupt load increase. However, lesser increases coordinated across many cores can have similar or worse effect.

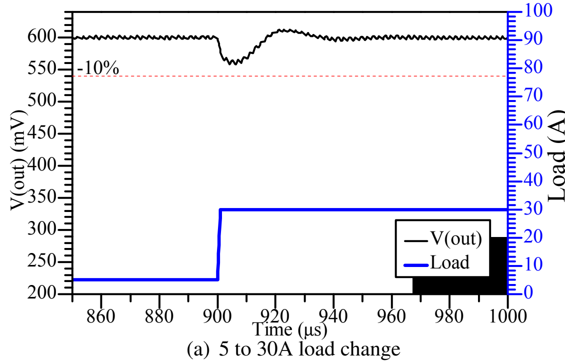
3. Voltage Droops in Multithreaded Workloads

Multithreaded workloads use synchronization primitives to coordinate activity in ways that can lead to simultaneous changes in compute intensity and power consumption.

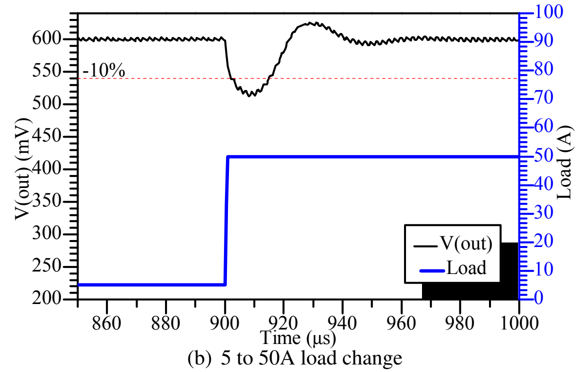
3.1. Barrier-Induced Droops

Barriers are particularly problematic because they typically coordinate the execution of large numbers of threads. Participating threads stall at a barrier until the last thread arrives. While threads are waiting to be released, their activity and power consumption are low. Moreover, power management techniques can clock-gate idling cores [1, 16] further reducing their power consumption. When idle threads are finally released from the barrier, cores typically experience a surge of activity, causing a spike in current demand and power consumption, which can lead to a voltage emergency.

Figure 2 shows the power consumption for a 4-core Intel Core i7 processor running the PARSEC benchmark *fluidanimate* with 4 working threads and the *sim-large* input set. The trace was obtained using Intel's Running Average Power Limit (RAPL) interface [8] that provides access to an internal energy counter updated every millisecond. The counter tallies the energy expended by the 4 cores but excludes the "un-core" components such as the on-chip memory controller. In addition to power, the figure also shows how many threads (cores) are blocked at a barrier at any given time. The figure captures one of the "frames" in the *fluidanimate* benchmark which includes eight barriers. In most cases, as cores arrive at a barrier, the power consumption drops followed by a significant spike. This is evident for all the barriers except the



(a) 5 to 30A load change



(b) 5 to 50A load change

Figure 1: Voltage regulator response to a small (a) and large (b) change in load.

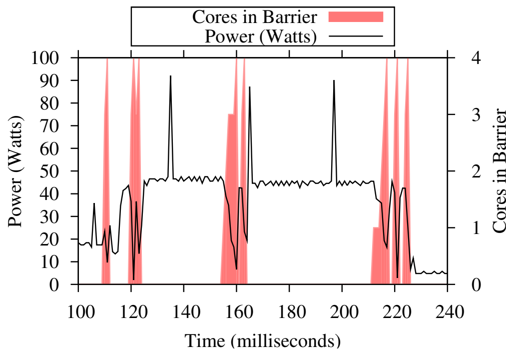


Figure 2: Processor power consumption while running the PARSEC benchmark *fluidanimate* on a 4-core Intel Core i7 system.

first and last ones, which occur during (or before) low-power serial sections. The power spikes are quite severe in some cases – for instance following barrier #5 – and can lead to voltage emergencies.

In this experiment synchronization events are not the only triggers of significant power fluctuations. We can observe other significant spikes that are likely caused by other architectural events such as long latency cache misses followed by bursts of activity. This behavior is consistent with that observed in previous work that examined voltage emergencies in 2-core [26] or 4-core [4] CMPs. Those studies did not single out synchronization events as a significant source of voltage emergencies.

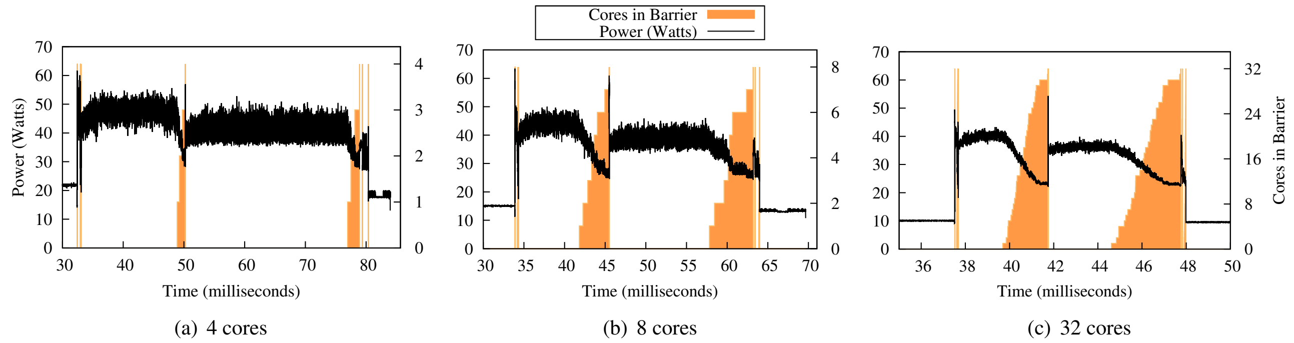
3.2. Impact of Core Count on Voltage Droops

Core counts are likely to continue to increase for the foreseeable future. We therefore examine the effects of synchronization events on power fluctuation in processors with larger numbers of cores. We run the same application (*fluidanimate*) on simulated many-core systems with 4, 8 and 32 cores, increasing the number of threads to match the number of cores. To provide a fair comparison of the relative magnitude of the voltage fluctuations, all systems are scaled to have the same maximum power (TDP). To keep simulation time reasonable, we use the *sim-small* reference input set. Details about the experimental methodology are provided in Section 5.

We find that the number of cores in the CMP has a direct impact on the magnitude of the power fluctuations caused by synchronization events, relative to those caused by core-level activity variation. Figure 3 shows the power profile for the three runs. Figure 3(a) shows that, for the 4-core case, the power profile follows that observed in the native execution (Figure 2). The simulation shows more local variability because the simulation models power at cycle granularity while the native execution can only be sampled every millisecond. Some of the power spikes correlate well with barrier activity. However, other local events are also causing significant fluctuations, just like in the native run. The ratio between the amplitude of the power spikes caused by barriers and non barrier-related events over the same time interval is close to 1, meaning they are equally likely to lead to emergencies.

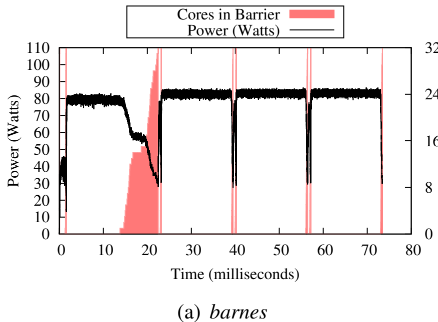
However, as the core count increases, within-core workload variability has a lower impact on chip-level power consumption. Events that trigger power fluctuations within individual cores are less likely to occur simultaneously on multiple cores and, as a result, their effects on power variability will tend to cancel each other out. This is visible in Figure 3(b), which shows power consumption when running on an 8-core configuration. Compared to the 4-core configuration, the benchmark exhibits less variability in power in sections of the application without barrier activity. However, around barriers, power fluctuations are much higher. The ratio between the barrier and non-barrier related spikes is closer to 3 in this case. This trend becomes significantly more pronounced for 32 cores (Figure 3(c)) with barrier-induced power spikes 6× larger than non barrier-related spikes. This trend suggests that, in the large CMPs of the near-future, coordinated activity fluctuation across many cores is much more likely to lead to voltage emergencies than within-core workload variability.

We have observed a similar behavior in most of the barrier-based benchmarks we examined. Figure 4 shows the power profile for *barnes* (a SPLASH2 application) running on 32 cores. *barnes* displays a very strong correlation



(a) 4 cores (b) 8 cores (c) 32 cores

Figure 3: Power variation for *fluidanimate* on CMP configurations with: (a) 4 cores, (b) 8 cores and (c) 32 cores.



(a) *barnes*

Figure 4: Power variation in response to barrier synchronization for *barnes*.

between barrier synchronization and variation in power consumption. For instance, as cores start to reach the first barrier, there is a gradual decrease in power consumption. When the barrier is exited, power consumption spikes, likely leading to a voltage emergency. Later barriers are entered a lot more rapidly by all threads, which causes sharp drops in power consumption, followed by spikes when they are released.

3.3. Other Voltage Droop-Causing Events

A condition signal broadcast has the potential to wake up many waiting threads simultaneously. We treat this as a special case of barrier synchronization, although none of our workloads have demonstrated this problem.

Lock synchronization does not generally cause the type of activity coordination that can lead to large power fluctuations. Even if a large number of threads are contending for a lock, they will acquire it sequentially, thus avoiding the activity spike typical for barriers. In our workloads we did not observe any emergencies that could be attributable directly to lock activity.

In addition to synchronization, some thread management functions can lead to large activity variation in parallel applications. These applications often create a set of worker threads during initialization, which are usually launched all at once. In this case, thread spawning can have an effect similar to barrier exit.

Active power management such as clock or power-gating can significantly reduce power consumption of idle threads. This can make voltage droops worse because the power fluctuation when transitioning into active state will generally be higher than if power management was not employed.

4. *VRSync* Design and Implementation

In order to eliminate synchronization induced voltage emergencies we develop *VRSync*, a novel synchronization methodology that controls core activity while in barriers and during barrier exit. We also apply *VRSync* to control the timing of thread spawning at the OS level.

4.1. Barrier Implementation

To reduce cache coherence traffic, we use a hierarchical barrier based on a binary software combining tree [20, 28]. In our implementation, a node is dynamically assigned to each participating thread. Each node has a sense flag that only its children observe while blocked on the barrier. The last thread to enter the barrier is assigned to the root node, at which time it inverts its sense flag to release waiting threads. Its children observe this and invert their sense flags, propagating this wake-up signal down the tree and releasing all threads from the barrier. The barrier tree is implemented as a one-dimensional array indexed by node numbers. This allows threads to locate their assigned node data structures in constant time, without traversing the tree. Array elements (including the sense flag variables) are allocated such that they will map to different cache lines, to avoid false sharing.

Threads are assigned nodes based on the order they enter the barrier (from highest numbered node to lowest). Node numbers are computed from a shared variable c, which is atomically incremented by each thread, starting from zero. Given a node number i, its parent node is $p = \lfloor (i-1)/2 \rfloor$, its left child is $l = 2i + 1$, and its right child is $r = 2i + 2$. With n cores participating in the barrier, $i = n - 1 - c$, so that for 32 cores, the first to enter the barrier is assigned node $i = 31$ and the last one is assigned node $i = 0$.

4.2. Scheduled Barrier Exit

VRSync implements a scheduled barrier exit to prevent the high current surges associated with barrier release. The schedule ramps up thread activity more slowly, leading to a more gradual rate of increase in power consumption. In *VRSync* threads participate in the barrier in two phases. The first is the *blocked* phase, prior to the time when all threads have arrived at the barrier. Cores assigned to blocked threads sleep or busy-wait in a way that minimizes power. Once all threads have arrived, threads enter the *delayed* phase. Cores assigned delayed threads continue to sleep until a thread-local timer has expired. We examine two exit schedules (*Linear* and *Bulk*) that control the pattern and rate at which cores are allowed to leave the *delayed* phase.

4.2.1 Linear Exit Schedule

The *Linear* schedule simply adds a fixed progressive delay to the wake-up of each thread that participates in a barrier. Figure 5(a) shows an example of the *Linear* exit schedule for 8 threads participating in two barriers. The first barrier phase proceeds as in a regular barrier. However, when the last thread reaches the barrier, all threads are released from the blocked phase (essentially clearing the barrier) and move into the *delayed* phase. The threads are scheduled to exit the *delayed* phase in the reverse order of their arrival. This is done as an optimization, to ensure that critical threads (those that arrive last at the barrier) will be released first. The assumption is that these threads are likely to remain critical for subsequent barriers, and giving them higher exit priority should improve performance. In Figure 5(a), thread T0 is the last to arrive at the first barrier and will be the first one to leave. The release of the rest of the threads is delayed by a `delay_unit` relative to the release of the previous thread. The value of the `delay_unit` is conservatively chosen to be the minimum the processor can tolerate without experiencing an emergency under worse-case load conditions.

Figure 5(c) illustrates the effect of the *Linear* schedule on the stability of the output voltage for a 32-core processor. It shows the output voltage over time, as cores are gradually turned on. This experiment assumes cores are initially off and will consume maximum power when on, although in reality, the power increase will be less. The *Linear* schedule gradually ramps up demand on the voltage regulator, which keeps the voltage droop above the safe margin of $\pm 10\%$. A more rapid ramp-up in demand would trigger a response similar to that in Figure 1(b), leading to an emergency.

4.2.2 Bulk Exit Schedule

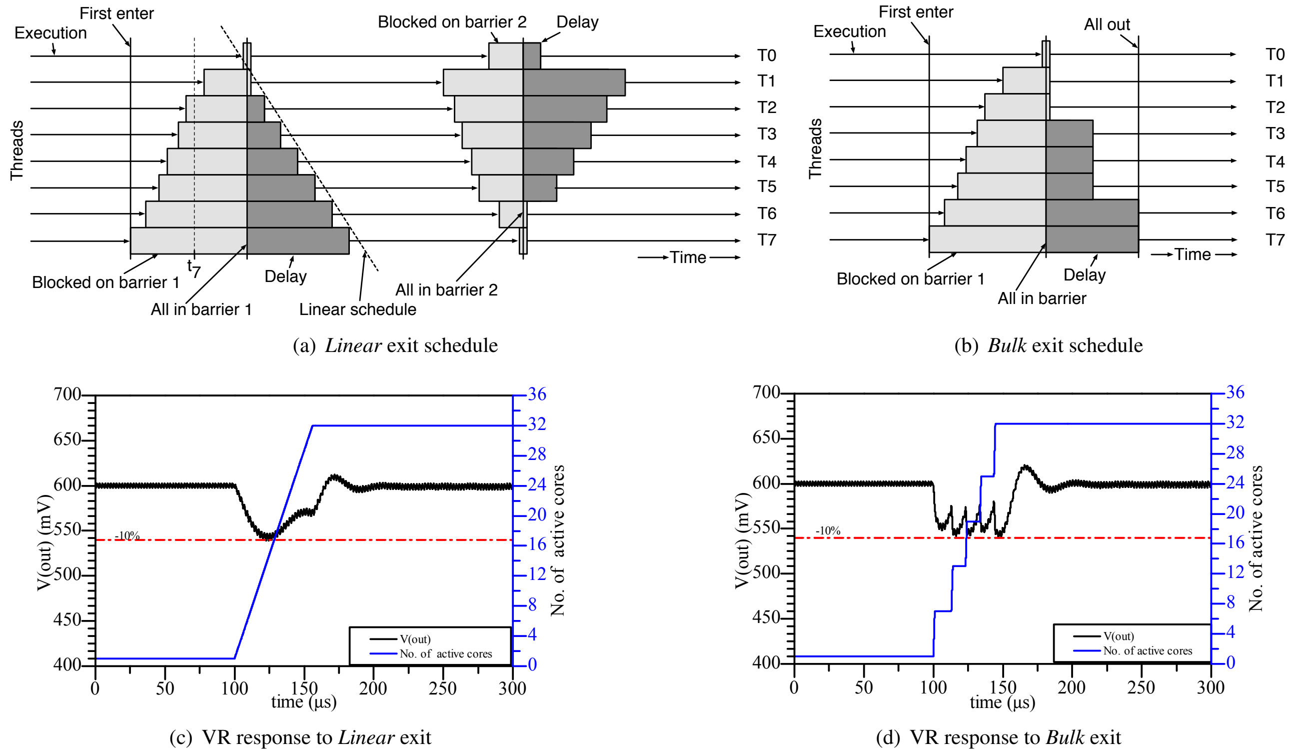
The *Linear* barrier exit schedule is relatively easy to test and implement. However, because voltage regulators have a non-linear response to transient workloads, it might be suboptimal. The speed of the regulator response depends on a number of factors including the quality of the control loop and the size of inductors and capacitors. The VR responds to a voltage droop by "pumping" additional current into the processor. The response however is not instantaneous. If the VR is given some time to catch up with the new demand, it might be able to respond faster to subsequent load increases.

We take advantage of the VR non-linearity with an alternative barrier exit schedule, called *Bulk*, which provides a faster exit compared to *Linear*. In the *Bulk* schedule, cores are released from the barrier in batches rather than one at a time. After a batch of cores is released, the VR is given some time to respond, followed by another batch, until all cores have exited the *delayed* phase of the barrier. Figure 5(b) shows an example with 8 threads leaving a barrier on the *Bulk* schedule. Figure 5(d) illustrates the VR response to a *Bulk* release of six cores at a time for a 32-core chip. Comparing Figures 5(c) and 5(d) we can see that the initial droop for the *Bulk* schedule is steeper but shorter, so it doesn't trigger an emergency. Because the load change is abrupt the VR responds aggressively to raise the voltage, which allows a second batch of six cores to be released without causing an emergency. Overall, the *Bulk* exit schedule completes about 20% faster than the *Linear* schedule for the same load.

4.3. Early Exit in Overlapping Barriers

Some applications make very heavy use of barrier synchronization, and their runtime could be hurt substantially by the delayed exit schedules. In heavily synchronized applications, barriers are often very closely spaced, with only a few instructions between them. As a result it is not uncommon for barriers to be "overlapping," with some threads exiting one barrier and entering a second before other threads have exited the first. *VRSync* makes overlapping barriers more likely because of the unequal delay it introduces in the exit time of each thread. A thread that enters the second barrier will rapidly go into a lower power state, reducing the load on the regulator. In this case, the scheduled barrier exit is unnecessarily conservative because it assumes all cores will consume peak power until the scheduled exit is complete. To eliminate the unnecessary overhead we would like to allow threads to exit the *delayed* phase of the barrier early. For instance, when a thread goes to block on the second barrier it could signal to another delayed thread that it can leave its *delayed* phase early.

Figure 6(a) shows an example of early exit applied to the *Linear* schedule. In this example thread T0 is last to reach the first barrier at time t_1 and will therefore be the first scheduled to exit. When T0 enters the second barrier at time t_2 it will trigger the early exit of thread T7 at time t_3. If there were no barrier overlap, T7 would have stayed in the *delayed* phase of the barrier until time t_5. The same pattern repeats until all threads exit the first barrier. This



(a) *Linear* exit schedule

(b) *Bulk* exit schedule

(c) VR response to *Linear* exit

(d) VR response to *Bulk* exit

Figure 5: Timing diagrams and VR response to the *Linear* exit schedule (a), (c) and the *Bulk* exit schedule (b), (d).

occurs much earlier (t_4) than the linear exit schedule would have dictated (t_5).

To implement the early exit schedule we add an `early_wake` variable to each node in the barrier tree. A thread entering the barrier will set this flag. If another thread is in the *delayed* phase on the same node, it will detect this flag change and exit immediately. Figure 6(b) shows the state of the barrier tree for the example in Figure 6(a), at time t_3. At that time, thread T0 has left the first barrier, arrived at the second barrier and been assigned to node 7. Before T0 goes into the blocked state, it wakes up thread T7, which was in *delayed* state at node 7. T7 goes on to block on node 6, waking up T6. At time t_3, threads T2 through T5 are in the *delayed* phase of the first barrier on nodes 2 to 5. Threads T1 and T6 are in execution between the two barriers and do not occupy any node in the tree at time t_3.

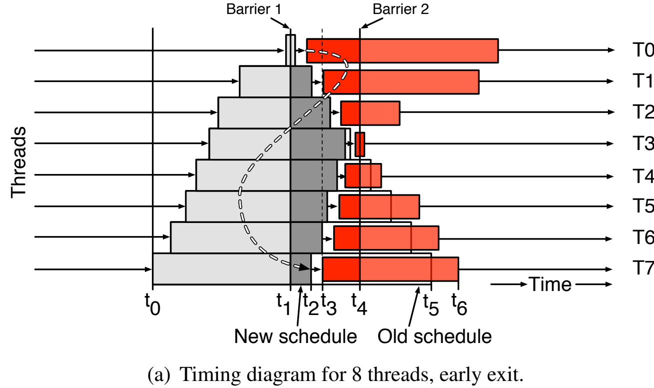
4.4. VRSync Implementation

VRSync is implemented as a user-level library that provides emergency-free synchronization and is installed system-wide. To implement the scheduled exit, *VRSync* requires a core-local timestamp, such as a cycle counter, which is used to time wake from the *delayed* barrier phase. The wake time is calculated, and a spin-wait executes until that time is reached. To implement spin-wait in a power-efficient manner the library uses a `PAUSE` instruction, like what was introduced to x86 processors in the SSE2 instruction set. On a `PAUSE` instruction, the CPU front-end inserts a long delay before fetching the next instruction, thereby significantly reducing the IPC and dynamic power of the core.

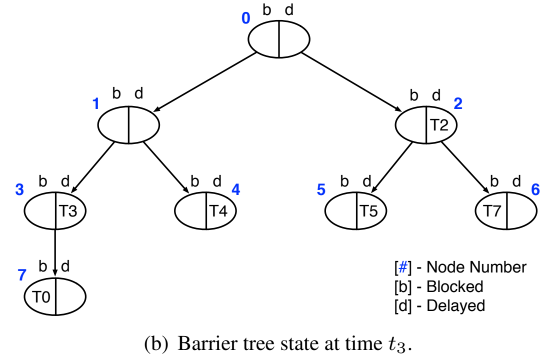
4.4.1 OS-Level *VRSync*

An OS-level *VRSync* implementation is also needed to prevent emergencies that can be caused by thread spawning or non-*VRSync* synchronization. These events involve OS interaction and must therefore be handled at OS-level. In essence, the OS must ensure that emergencies are not triggered when work is scheduled to cores that have been previously idle or sleeping. New work is assigned to idle cores by issuing an interrupt to these cores. The interrupt can come from another thread (spawn) or from a hardware device (I/O). *VRSync* augments this mechanism by scheduling core wake-up according to the *Linear* or *Bulk* schedules, avoiding emergencies.

The same OS-level implementation ensures that applications using standard synchronization instead of the *VRSync* library will not trigger voltage emergencies. Most barrier primitives use blocking calls handled by the OS. When threads are released from a standard barrier, they appear to the OS as core-wakeup events. Since these are scheduled by the OS-level *VRSync*, they will not cause emergencies. However, users will have an incentive to use the *VRSync* library instead of standard synchronization for performance reasons. The OS implementation of *VRSync* has a higher



(a) Timing diagram for 8 threads, early exit.



(b) Barrier tree state at time t_3.

Figure 6: Example of early exit from the *Linear* schedule due to overlapping barriers.

overhead than the library, and it cannot take advantage of performance optimizations like early exit for overlapping barriers. As we show in Section 6, these optimizations can reduce the overhead of *VRSync* by as much as 25×.

4.4.2 *VRSync* Design Parameters

The parameters chosen for the *Linear* and *Bulk* exit schedules of *VRSync* are dependent on the regulator design, size of capacitors, number of regulator phases, etc. They are also dependent on the desired safety margins and the power characteristics of the cores. These parameters would therefore need to be determined by processor manufacturers. Since processors and voltage regulators are often developed by different manufacturers, data from regulator data sheets or manufacturer-supplied models can be used to perform the necessary design-time simulations. Once the parameters are chosen, they can be programmed in the system firmware and accessed by the synchronization libraries.

To determine these design parameters, we simulate a commercial regulator from Linear Technologies [17] using LTspice [18] and a manufacturer supplied model. We measure regulator response to load changes, assuming worst-case power consumption for each core. For the *Linear* schedule, multiple simulations are run with different delay values until the minimum value that does not cause an emergency is found. Similarly, for *Bulk* we determine the optimal number of cores that can be released together and the delay between bulk exits.

5. Evaluation Methodology

5.1. Architectural Simulation Setup

Most of our experiments are conducted on a simulated 32-core CMP in 32nm technology using SESC [27]. Each core is a dual-issue out-of-order architecture. SESC was modified to run a port of the *LinuxThreads* library, a simple implementation of *POSIX Threads* required by the PARSEC benchmarks. We collected runtime and energy information. Table 1 summarizes the architecture configuration.

CMP architecture	
Cores	32, out-of-order
Fetch/issue/commit width	2/2/2
Register file size	76 int, 56 fp
Instruction window	56 int, 24 fp
L1 data cache	4-way 16KB, 2-cycle
L1 instruction cache	2-way 16KB, 2-cycle
Private L2	8-way 256KB, 10-cycle
NoC Interconnect	2D Torus
Coherence	L2-level, MESI
Technology	22nm
V_{dd}	600mV
Clock Frequency	1GHz

Table 1: Summary of the experimental parameters.

We ran the PARSEC benchmarks *blackscholes, bodytrack, fluidanimate, swaptions, dedup,* and *streamcluster* and the SPLASH2 benchmarks *barnes, cholesky, fft, lu, ocean, radiosity, radix, raytrace,* and *water-nsquared* with the reference, sim-small input sets. The benchmark sets include applications with light, moderate and very heavy barrier activity as well as applications that use no barriers. Most applications use at least some lock synchronization and some of the PARSEC benchmarks use condition waits.

To model power consumption at low voltage we used the models from Markovìc et al. [19]. These were integrated into CACTI [23] to extract energy per access for all the SRAM memory structures including register file, caches, etc. The low voltage models were also used to scale the existing SESC power model for logic units. To model NoC power we used Orion 2 [14].

5.2. Voltage Regulator Simulation Setup

For this work, we used a state of the art voltage regulator, Linear Technology's LTC3729L-6 polyphase, synchronous step-down switching regulator [17]. This is a commercially-available regulator intended for use in desktop computers and servers. The regulator's behavior is simulated in LTspice

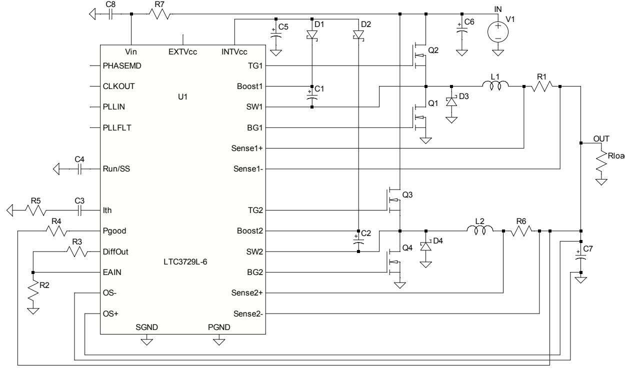


Figure 7: Diagram of the voltage regulator circuit design (two of the six phases).

IV [18] using a model provided by the manufacturer. We set up the LTC3729L-6 in a 6 phase design, which requires 3 LTC3729L-6 chips with 12 onboard capacitors of 270 µF each and a combined ESR of $2\,\mathrm{mOhms}$. The controllers are synchronized using the CLKOUT pin of the first chip. Figure 7 shows the circuit diagram for two of the regulator phases. The input voltage was set at 12V and the output voltage set at 600mV by means of a resistor divider circuit using remotely sensed output voltage. The regulator configuration and capacitor values were chosen to provide the maximum current required ($160\ \mathrm{Amps}$) for low-Vdd operation, in line with Intel specifications [10]. The regulator drives a current sink that models the processor load.

The current traces obtained from the SESC simulator were fed into the regulator to measure the regulator response to a wide range of load steps and various load slews. We measured the regulator response to a sweep of current changes and identified the maximum rate at which load can change without causing an emergency (Max dI/dt). We define a voltage emergency as occurring if the output voltage droops below 10% of the target voltage. While this margin is consistent with industry practices and previous work, it is the result of a tradeoff between power consumption goals, component (e.g. capacitors, controllers) cost and size, etc.

We use Max dI/dt to identify emergencies in the power consumption profile extracted from the microarchitectural simulator. Changes in current that exceed Max dI/dt are flagged as emergencies. Parameters for the *Linear* and *Bulk* exit schedules are determined using VR simulations, based on worst-case assumptions about power consumption. Once delay schedules are determined, they are programmed into our *VRSync* implementation in the simulator.

6. Evaluation

In this section we analyze the incidence of voltage emergencies across different multi-threaded applications and the effectiveness of *VRSync* at eliminating them. We also evaluate the impact of the *VRSync Linear* and *Bulk* policies on execution time and energy.

The evaluation includes two baseline systems. The more conservative baseline uses no active power management, meaning that when blocked on a barrier, threads will simply issue a `PAUSE` instruction that will reduce the IPC and power of the busy-wait loop. The second baseline employs active power management through clock-gating at core level. When blocked, threads execute a `HALT` instruction, which cuts the core's dynamic power.

6.1. Voltage Emergencies

We analyze power traces obtained from simulator runs to identify voltage emergencies. These are defined as voltage droops larger than 10%, which would be triggered by current changes that exceed Max dI/dt. Table 2 shows the results of this study. For each benchmark we show both the number of dynamic barriers and the number of emergencies for the two baseline runs. In general, the number of emergencies correlates very well with the number of barriers. For instance *ocean* has a very large number of barriers and also experiences the largest number of voltage emergencies. This correlation however doesn't always hold. *streamcluster* has extremely heavy barrier activity, yet it registers no emergencies. This is likely due to the fact that *streamcluster* has relatively low IPC and low maximum power consumption. As a result, when threads exit from barriers their activity does not increase sufficiently to cause emergencies.

Table 2 also shows the number of emergencies that occur in the baseline system that uses clock gating. The number of emergencies increases for most benchmarks. This is because cores idle with lower power while blocked at a barrier, leading to higher power spikes upon exit. For *ocean* the number of emergencies increases by almost 30%.

6.1.1 Eliminating Emergencies with *VRSync*

Figure 8(a) shows an example of a barrier that leads to an emergency in *fluidanimate*. We show power consumption over time, the number of threads in a barrier at any given time and the location of emergencies (red arrow, at the top of the graph). We can see that as threads enter the barrier, power consumption gradually drops, followed by a big spike upon exit. Soon after, threads enter a second barrier, but this one does not lead to an emergency.

Figure 8(b) shows the effect of the *Linear* schedule on the same section of the benchmark. This schedule leads to a more gradual resumption of compute activity, which eliminates the emergency. This example illustrates the effects of the overlapping barrier optimization which allows early exit from the first barrier as threads enter the second one.

Figure 8(c) shows the effects of the *Bulk* schedule on the same code. We can see that the exit from the barrier now occurs in batches of threads, leading to a step-wise, but still

Benchmark	*Num. of* barriers	*Number of emergencies*			
		Baseline	*Baseline w/ clock gating*	*VRSync Linear*	*VRSync Bulk*
radiosity	10	0	1	0	0
barnes	17	0	0	0	0
ocean	900	419	543	0	0
raytrace	1	0	0	0	0
water-nsq'd	20	7	10	0	0
cholesky	4	0	0	0	0
fft	7	8	8	0	0
lu	67	50	56	0	0
radix	11	2	3	0	0
blackscholes	1	0	0	0	0
bodytrack	80	0	0	0	0
fluidanimate	24	9	10	0	0
swaptions	0	9	9	0	0
dedup	0	0	0	0	0
streamcluster	4396	0	0	0	0

Table 2: Number of barriers and number of emergencies for the baseline system, the baseline with clock gating and for *VRSync Linear* and *Bulk*. *VRSync* eliminates all emergencies for clock-gated and non-clock-gated cases.

gradual ramp-up in power. Again, the emergency is avoided. The *Bulk* exit is also faster than the *Linear* one, but both are slower than the baseline.

Figure 9 shows the power profiles for *lu*, *fft* and *swaptions*. Figure 9(a) shows *lu* running on the baseline with no clock gating. For *lu*, emergencies are perfectly correlated with barrier exits. Because of high activity following the barriers, almost all barrier exits lead to emergencies. Figure 9(d) shows that the *Bulk* schedule eliminates all emergencies with a slight increase in execution time.

In some cases, program phases can remain highly synchronized long after the synchronization event has completed. This occurs in parallel workloads with very balanced workload distribution across threads. An example of this behavior can be seen in Figure 9(b), which shows power consumption for *fft*. This workload is characterized by alternating periods of high and low IPC (due to phases of low and high last-level cache misses), and these periods occur nearly simultaneously across all threads, even though no synchronization is present. These synchronous fluctuations lead to multiple emergencies. Figure 9(e) shows how the *Linear* exit schedule significantly diminishes these power fluctuations. This happens because the *Linear* release schedule realigns threads relative to each other leading to less overlap in the high and low activity phases, completely eliminating emergencies from *fft*.

All emergencies in *swaptions* are caused by simultaneous spawning of large numbers of threads. Figure 9(c) shows a short section of the application startup. Several emergencies occur early on in the execution. All of these are eliminated by the *VRSync* scheduled thread spawn as Figure 9(f) shows.

Benchmark	*Linear*	*No overlap Linear*	*Bulk*	*No overlap Bulk*
ocean	1.62	3.36	1.50	2.96
streamcluster	2.10	24.90	1.36	16.75
g.mean (ALL)	1.11	1.39	1.06	1.35

Table 3: Runtimes (relative to baseline) for *ocean*, *streamcluster*, and the geometric mean over all benchmarks. For these benchmarks, the overlapping barrier optimization is critical for good performance.

6.2. *VRSync* Impact on Execution Time

VRSync delayed exit can impact execution time because it forces threads to spend additional time in barriers. Figure 10 shows the execution time of all benchmarks for the *Linear* and *Bulk* exit schedules with and without the early exit optimization.

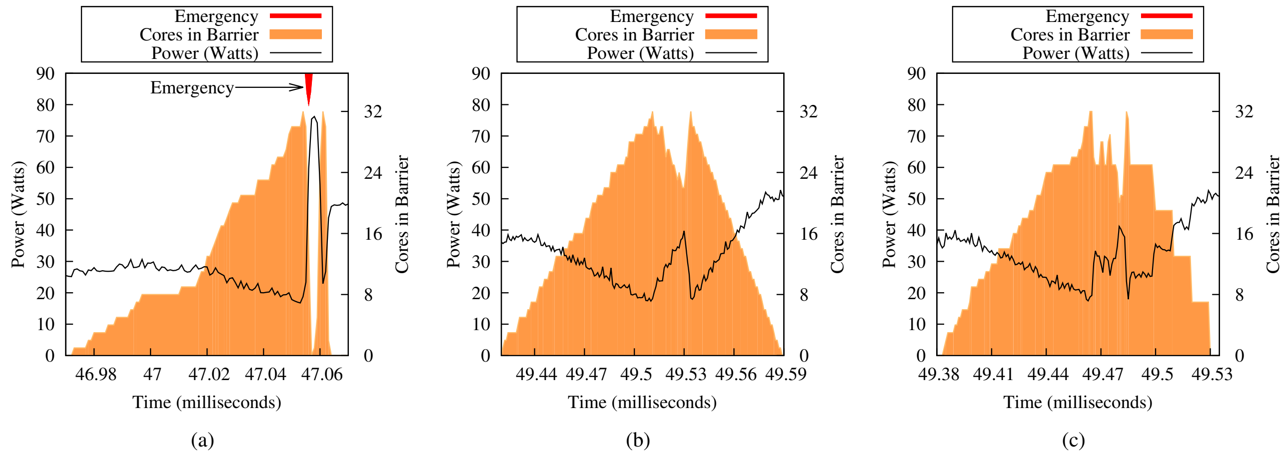
As expected, the *Linear* schedule has the highest overhead, with an average increase in execution time of 11% across all the benchmarks. Applications that have moderate to no barrier activity have very small increase in runtime, between 0 and 10%. *fft* is an exception because, even though it has few barriers, its runtime is very short, making barrier exit schedules a significant fraction of its runtime.

Applications with heavy barrier activity suffer significantly more from *VRSync*. *streamcluster* has by far the highest overhead, with a 2.1× increase in execution time. The high overhead is due to the very large number of barriers (4396) used by this benchmark. *ocean* has the second largest overhead (62%), again because of the large number of barriers (900).

The *Bulk* schedule reduces the average execution time overhead to 6.3%. *Streamcluster* shows a dramatic improvement with just over 36% overhead compared to 100% with the *Linear* schedule. This is due to the interplay between the exit schedule and the early exit optimization. The *Bulk* schedule releases multiple threads right at the barrier exit, which quickly reach a new barrier, triggering early exit sooner than in the *Linear* case. Overlapping barriers with early exit are very common in *streamcluster*.

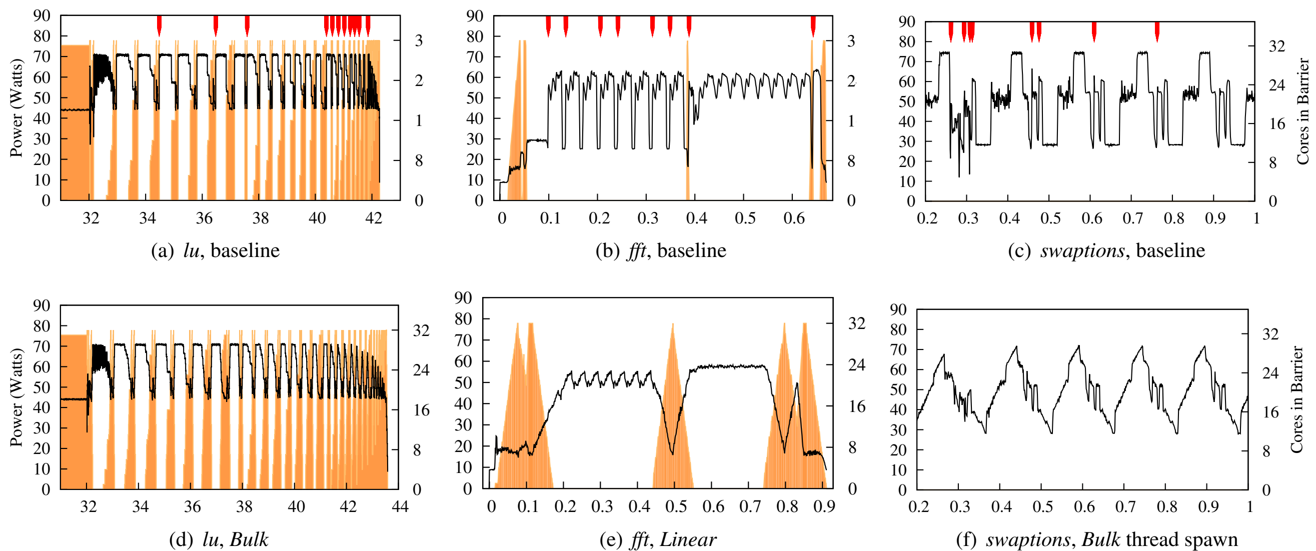
The early exit optimization has a very big impact on the performance of *VRSync* in benchmarks with large numbers of barriers. As Table 3 shows, without early exit, the runtime overhead would be as high as 3.36× for *ocean* and as high as 24.9× for *streamcluster*. The average runtime overhead for *Bulk* without early exit would be close to 35% for instead of 6.3% with early exit.

Barnes, raytrace, and *dedup* actually speed up slightly. When parallel tasks simultaneously go through periods of high memory activity, they compete for memory bandwidth and slow each other down. *VRSync* shifts the alignment of those phases, reducing competition for shared resources and making execution more efficient.



(a) (b) (c)

Figure 8: Power variation in response to synchronization for a barrier from *fluidanimate*: (a) baseline without clock gating (b) *Linear* barrier exit schedule and (c) *Bulk* exit schedule.



(a) *lu*, baseline (b) *fft*, baseline (c) *swaptions*, baseline

(d) *lu*, *Bulk* (e) *fft*, *Linear* (f) *swaptions*, *Bulk* thread spawn

Figure 9: Power profile for *lu*, *fft* and *swaptions* for the baseline without clock gating (a), (b), (c), for the *Bulk* (d), and *Linear* (e) schedules, and for scheduled spawn only (f).

6.3. *VRSync* Energy

We examine the energy implications of using the proposed VR-aware scheduling versus other options for avoiding voltage emergencies. Table 4 summarizes the results. Each entry in the table shows the average runtime, power and energy for the different techniques relative to a baseline without clock gating. Because the baseline has no barrier exit scheduling and a small guardband, it cannot avoid emergencies.

Another option for eliminating voltage emergencies is to increase the voltage guardband, while using faster versions of *VRSync*. *Bulk* $160mV$ shows a combination of higher guardband with the *Bulk* scheduling policy. The voltage guardband for the this option increases from $60mV$ to $160mV$. This allows for a faster exit schedule that the regular *Bulk* schedule, reducing the runtime overhead. However, because the supply voltage is higher, energy consumption increases by 42%. Note that even with the higher guardband *VRSync* is still needed to guarantee emergency-free execution.

We also define a guardband-only option, which we call *Optimistic guardband*, based on our workloads. We identified the steepest dI/dt of any of our benchmarks and determined the necessary safe voltage guardband to avoid emergencies without *VRSync*. The *Optimistic guardband* case would increase energy by 56% over baseline.

From the summary in Table 4 we can see that the most energy efficient option for avoiding voltage emergencies is *VRSync Bulk* with only 4.9% increase in energy over a

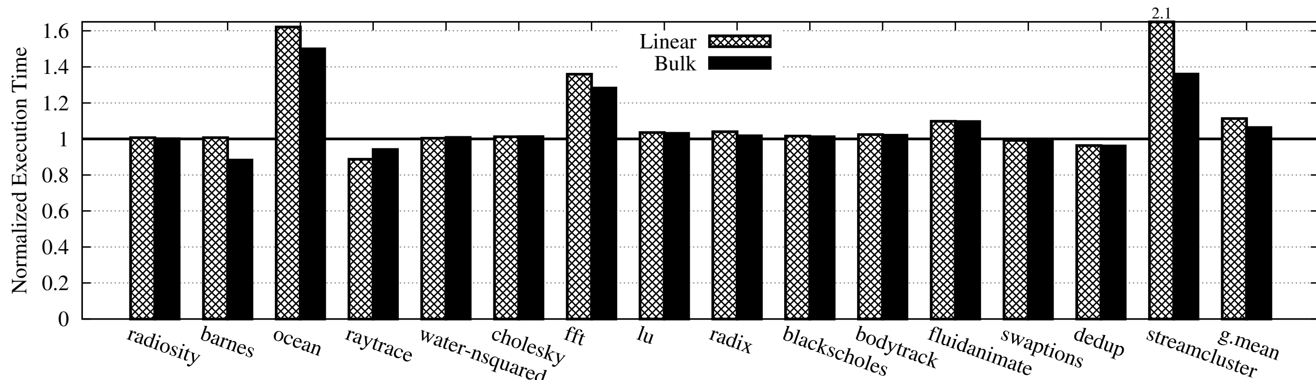


Figure 10: *VRSync* execution times for *Linear* and *Bulk* schedules, normalized to baseline.

Technique	*Guardband*	*Schedule*	*Emergencies*	*Runtime*	*Power*	*Energy*
Baseline	60mV	(None)	yes	1.0	1.0	1.0
VRSync Linear	60mV	Linear	no	1.112	0.98	1.086
VRSync Bulk	60mV	Bulk	no	1.063	0.99	1.049
Bulk 160mV	160mV	Bulk	no	1.045	1.361	1.422
Optimistic guardband	210mV	(None)	no	1.0	1.563	1.563

Table 4: The effects of different guardbands on average benchmark execution time, power, energy, and emergencies.

baseline with emergencies. This solution also uses only 67% of the energy of a CMP with the *Optimistic guardband* needed to eliminate voltage emergencies.

7. Related Work

Prior work has focused on addressing voltage emergencies in low core count systems. Reddi et al. [25] proposed a solution for eliminating emergencies in single-core CPUs. They employ heuristics and a learning mechanism to predict voltage emergencies from architectural events. When an emergency is predicted, execution rate is throttled, reducing the slope of current changes. Gupta et al. [7] proposed an event guided adaptive voltage emergency avoidance scheme: Recurring emergencies are avoided by initiating various operations such as *pseudo-nops*, prefetching, and a hardware throttling mechanism on events that cause emergencies. Gupta et al. also proposed DeCoR [6], a checkpoint/rollback solution which allows voltage emergencies but delays the commit of instructions until they are considered safe. A low voltage sensor, of known delay, signals that an emergency is likely to have occurred and the pipeline is flushed and rolled back to a safe state.

Powell and Vijaykumar [24] proposed two approaches for reducing high-frequency inductive noise caused by processor pipeline activity. Pipeline muffling reduces the number of functional units switching at any given time by controlling instruction issue. A priory current ramping slowly ramps up the current of functional units before they are utilized in order to reduce the amplitude of the current surge.

A software approach to mitigating voltage emergencies was proposed by Gupta et al. in [5]. They observe that a few loops in SPEC benchmarks are responsible for the majority of emergencies in superscalar processors. Their solution involves a set of compiler-based optimizations that reduce or eliminate architectural events likely to lead to emergencies such as cache or TLB misses and other long-latency stalls.

Very few previous studies have examined voltage emergencies in multicore chips. Gupta et al. [4] characterize within-die voltage variation using a detailed distributed model of the on-chip power-supply grid. They model a 4-core CMP and use a multi-programmed workload consisting of SPEC applications in their evaluation. Reddi et al. [26] evaluate voltage droops in an existing dual-core microprocessor. They propose designing voltage margins for typical instead of worst-case behavior, relying on resilience mechanisms to recover from occasional errors. They also propose co-scheduling threads with complementary noise behavior, to reduce voltage droops. We are not aware of any previous work that examines voltage emergencies in CMPs with large numbers of cores running multithreaded applications as we do in this work.

A significant amount of recent work has demonstrated dramatic power savings from low voltage operation. These works aggressively scale supply voltage to very close to the threshold voltage [19, 2, 3, 21, 22]. In general, two orders of magnitude power savings are possible with an order of magnitude reduction in frequency. Overall the technology promises an order of magnitude reduction in energy. Many challenges remain, including reliability and high variation. Achieving low voltage operation requires significant reduction in voltage margins that can only be achieved if other techniques for reducing voltage droops are developed.

8. Conclusion and Future Work

Market and technology factors are driving CPU architects to employ increasingly aggressive energy and power-saving design techniques. Lowering supply voltage makes chips more susceptible to the effects of severe supply voltage fluctuations, which can lead to errors. In this paper we identify an important problem that will challenge future low-voltage

CMPs. We show that in large systems, voltage emergencies will be caused almost exclusively by coordinated activities across many cores, such as barrier synchronization, where multiple cores experience sudden changes in compute demand simultaneously. We propose a set of low overhead and highly effective techniques for mitigating these challenges. Unlike previous work, our solutions are deterministic and do not rely on heuristics for predicting when voltage emergencies are likely to occur.

We hope this paper will inspire future research on this topic. One aspect we would like to address is the impact of multiple voltage domains on voltage emergencies. Future CMPs with hundreds of cores are likely to have cores organized into clusters, with each cluster receiving an independently regulated voltage supply. Voltage droops caused by cross-domain workload migration and other aspects related to multiple voltage domains will have to be investigated. In this work we showed how clock gating can be a significant source of voltage droops. In future work we will investigate the impact of other power management techniques, such as core-level power gating, on voltage stability.

References

[1] A. Bhattacharjee and M. Martonosi. Thread criticality predictors for dynamic performance, power, and resource management in chip multiprocessors. In *International Symposium on Computer Architecture*, pages 290–301, June 2009.

[2] L. Chang, D. Frank, R. Montoye, S. Koester, B. Ji, P. Coteus, R. Dennard, and W. Haensch. Practical strategies for power-efficient computing technologies. *Proceedings of the IEEE*, 98(2):215–236, February 2010.

[3] R. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge. Near-threshold computing: Reclaiming Moore's law through energy efficient integrated circuits. *Proceedings of the IEEE*, 98(2):253–266, February 2010.

[4] M. S. Gupta, J. Oatley, R. Joseph, G.-Y. Wei, and D. Brooks. Understanding voltage variations in chip multiprocessors using a distributed power-delivery network. In *Design Automation and Test in Europe*, pages 1–6, April 2007.

[5] M. S. Gupta, K. K. Rangan, M. D. Smith, G.-Y. Wei, and D. Brooks. Towards a software approach to mitigate voltage emergencies. In *International Symposium on Low Power Electronics and Design*, pages 123–128, August 2007.

[6] M. S. Gupta, K. K. Rangan, M. D. Smith, G.-Y. Wei, and D. Brooks. Decor: A delayed commit and rollback mechanism for handling inductive noise in processors. In *IEEE International Symposium on High-Performance Computer Architecture*, pages 381–392, February 2008.

[7] M. S. Gupta, V. Reddi, G. Holloway, G.-Y. Wei, and D. Brooks. An event-guided approach to reducing voltage noise in processors. In *Design Automation and Test in Europe*, pages 160–165, April 2009.

[8] Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3, Section 14.7 (as of November 2011).

[9] Intel Press Release. Intel 22nm 3-D tri-gate transistor technology, May 2011.

[10] Voltage regulator module (VRM) and enterprise voltage regulator-down (EVRD) 11.1 design guidelines. Technical report, Intel Corp., September 2009.

[11] International Technology Roadmap for Semiconductors (2009).

[12] N. James, P. Restle, J. Friedrich, B. Huott, and B. McCredie. Comparison of split-versus connected-core supplies in the power6 microprocessor. In *International Solid-State Circuits Conference*, pages 298–604, February 2007.

[13] R. Joseph, D. Brooks, and M. Martonosi. Control techniques to eliminate voltage emergencies in high performance processors. In *IEEE International Symposium on High-Performance Computer Architecture*, pages 79–90, 2003.

[14] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi. ORION 2.0: A fast and accurate NoC power and area model for early-stage design space exploration. In *Design Automation and Test in Europe*, pages 423–428, 2009.

[15] J. Kavalieros, B. Doyle, S. Datta, G. Dewey, M. Doczy, B. Jin, D. Lionberger, M. Metz, W. Rachmady, M. Radosavljevic, U. Shah, N. Zelick, and R. Chau. Tri-gate transistor architecture with high-k gate dielectrics, metal gates and strain engineering. In *Proceedings of the Symposium on VLSI Technology, Digest of Technical Papers*, pages 50–51, 2006.

[16] J. Li, J. Martínez, and M. C. Huang. The thrifty barrier: Energy-aware synchronization in shared-memory multiprocessors. In *IEEE International Symposium on High-Performance Computer Architecture*, pages 14–24, February 2004.

[17] Linear Technologies, LTC3729L. `http://www.linear.com/product/LTC3729L-6`.

[18] Linear technologies, LTSpice. `http://www.linear.com/designtools/software/LTspice`.

[19] D. Markovic, C. Wang, L. Alarcon, T.-T. Liu, and J. Rabaey. Ultralow-power design in near-threshold region. *Proceedings of the IEEE*, 98(2):237–252, February 2010.

[20] J. Mellor-Crummey and M. Scott. Algorithms for scalable synchronization on shared-memory multiprocessors. *ACM Transactions on Computer Systems (TOCS)*, 9(1):21–65, 1991.

[21] T. Miller, J. Dinan, R. Thomas, B. Adcock, and R. Teodorescu. Parichute: Generalized turbocode-based error correction for near-threshold caches. In *International Symposium on Microarchitecture*, pages 351–362, December 2010.

[22] T. Miller, R. Thomas, X. Pan, N. Sedaghati, and R. Teodorescu. Booster: Reactive core acceleration for mitigating the effects of process variation and application imbalance in low-voltage chips. In *IEEE International Symposium on High-Performance Computer Architecture*, pages 27–38, February 2012.

[23] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi. CACTI 6.0: A tool to model large caches. Technical Report HPL-2009-85, HP Labs, 2009.

[24] M. D. Powell and T. N. Vijaykumar. Pipeline muffling and a priori current ramping: architectural techniques to reduce high-frequency inductive noise. In *International Symposium on Low Power Electronics and Design*, pages 223–228, August 2003.

[25] V. J. Reddi, M. S. Gupta, G. Holloway, G. yeon Wei, M. D. Smith, and D. Brooks. Voltage emergency prediction: Using signatures to reduce operating margins. In *IEEE International Symposium on High-Performance Computer Architecture*, 2009.

[26] V. J. Reddi, S. Kanev, W. Kim, S. Campanoni, M. D. Smith, G.-Y. Wei, and D. Brooks. Voltage smoothing: Characterizing and mitigating voltage noise in production processors via software-guided thread scheduling. In *International Symposium on Microarchitecture*, pages 77–88. IEEE Computer Society, 2010.

[27] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, K. Strauss, S. Sarangi, P. Sack, and P. Montesinos. SESC Simulator, January 2005. http://sesc.sourceforge.net.

[28] P.-C. Yew, N.-F. Tzeng, and D. Lawrie. Distributing hot-spot addressing in large-scale multiprocessors. *IEEE Transactions on Computers*, C-36(4):388–395, April 1987.