LeadOut: Composing Low-Overhead Frequency-Enhancing Techniques for Single-Thread Performance in Configurable Multicores*

Brian Greskamp Ulya R. Karpuzcu Josep Torrellas
Department of Computer Science
University of Illinois at Urbana-Champaign
http://iacoma.cs.uiuc.edu

Abstract

Despite the ubiquity of multicores, it is as important as ever to deliver *high single-thread performance*. An appealing way to accomplish this is by shutting down the idle cores in the chip and running the busy, performance-critical core(s) at higher-than-nominal frequencies. To enable such frequencies, two low-overhead approaches either boost voltage beyond nominal values, or pair cores in leader-checker configurations and let them run beyond safe frequency margins.

We observe that, in a large multicore with varying numbers of busy cores, individual application of either of these two techniques is suboptimal. Each alone is often unable to bring the multicore all the way to its power or temperature envelopes due to limitations in supply voltage or error rate. Moreover, we show that the two techniques are complementary, and can be synergistically combined to unlock much higher levels of single-thread performance. Finally, we demonstrate a dynamic controller that optimizes the two techniques. Our data shows that, given a 16-core multicore where half of the cores are already busy, an additional, performance-critical thread now attains 34% higher performance than before, while consuming 220% more power.

1 Introduction

Although multicores are becoming the commodity architecture for a wide range of platforms, it is as important as ever for processors to deliver *high single-thread performance*. To tackle this problem, many techniques have been proposed, such as thread-level speculation (e.g., [10, 19, 28, 29]) or core fusion/composition [16, 17]. To be attractive, these techniques have to deliver high single-thread performance enhancements with modest hardware complexity.

Often, some of the cores in a multicore are idle. Consequently, an appealing approach to improve per-thread performance is to shut down the idle cores in the chip and run the busy, performance-critical core(s) at higher-than-nominal frequencies. To enable such frequencies, two relatively low-complexity techniques are: either boost the voltage beyond its nominal value, or pair cores in leader-checker configurations and let them run beyond safe margins.

The first technique is included in Intel's Nehalem processor as Turbo Boost [14]. The idea is to increase the voltage and frequency of the active cores beyond the values corresponding to nominal operating conditions until the chip reaches the extremes of its power envelope. The hardware overhead is modest because it uses similar mechanisms as dynamic voltage-frequency scaling (DVFS), which is already widely deployed for power savings. Further, we expect that future power-constrained multicores will include flexible DVFS capabilities such as several voltage and frequency domains in order to maximize power efficiency.

The second technique is configurable timing speculation using a leader–checker core pair. The idea, as implemented in the Paceline microarchitecture [9], follows the Slipstream design [30] where the same thread runs on two nearby cores. The leader core is overclocked to the point of suffering timing errors, while the checker core runs at a safe frequency and periodically checks (and corrects) the leader. The checker keeps up because the leader supplies branch predictions and brings data into an L2 cache shared by the two cores. Overall, the two cores effectively run an unmodified thread at a higher frequency. The key hardware changes involve passing branch predictions and comparing register checkpoints, leaving the core mostly unmodified.

As multicores scale to more cores, they will often run a mixed workload that leaves some cores idle, uses some set of cores for throughput, and demands maximum sequential performance from a few latency-critical threads. Since these workloads will be dynamic, the number of throughput-oriented and latency-critical threads will vary over time and from application to application. In such

*This work was supported by Sun Microsystems under the UIUC OpenSPARC Center of Excellence, NSF under grant CPA-0702501, and SRC GRC under grant 2007-HJ-1592.

an environment, this paper observes that individual application of either voltage–frequency boosting or leader–checker execution is suboptimal. Alone, neither technique will be able to consistently bring the multicore to the extremes of its power envelope or to its limit temperature. Instead, each individual technique will be limited by other factors, such as the maximum supply voltage (beyond which reliability suffers) or maximum error rate (beyond which performance drops due to frequent recovery), leaving some of the available power and temperature headroom untapped.

This paper shows that the two techniques are in fact orthogonal and can combine synergistically to unlock much higher levels of single-thread performance than either can alone. Moreover, we demonstrate a dynamic controller that optimizes the two techniques. Under a variety of loading conditions, the two techniques together bring the multicore to the extremes of its power and temperature envelope.

Finally, this paper evaluates the controlled combination of these techniques on a 16-core configurable multicore where each core has its own voltage and frequency domain. Our results show that, when half of the cores are busy and the goal is to optimize an additional, performance-critical thread, combining these techniques enables the thread to attain 34% higher performance than it had before, while consuming 220% more power. Moreover, the configurability of the proposed multicore enables similar gains under various load conditions.

This paper is organized as follows: Section 2 gives a background; Section 3 shows that the two techniques for single-thread performance are synergistic; Section 4 describes a practical control system to manage the techniques; Sections 5 and 6 evaluate the system; and Section 7 discusses related work.

2 Background

This section reviews the two techniques to support higher-than-nominal frequency.

2.1 Boosting Voltage Beyond Nominal Point

The Intel Core i7 ("Nehalem") processor introduces the Intel Turbo Boost technology, where both supply voltage (V_{dd}) and frequency (f) are increased when high performance is desired [14]. To support this feature, the processor includes sensors and an on-chip controller that continuously monitors the power and temperature in the chip.

When the operating system requests performance state P0 (highest performance) for one or more cores and places other cores into an inactive state, the on-chip controller transparently attempts to scale up V_{dd} and f on the active cores. As long as the on-chip sensors report safe conditions, the controller increases the frequency in steps of 133.33 MHz with a corresponding voltage increase. The maximum frequency increase (maximum number of steps) is determined by the number of inactive cores. If, at any time, the power and temperature conditions reach unsafe values, the hardware automatically decreases V_{dd} and f one 133.33 MHz step.

Note that in Intel's Turbo Boost, all active cores increase and decrease V_{dd} and f at the same time. In this paper, we use a more advanced design where each core can change its V_{dd} and f independently of the other cores. Per-core voltage domains can be implemented with multiple on-die [18] or external regulators. Current systems already include per-core frequency domains [6].

2.2 Leader–Checker Core Pairing

In a leader-checker microarchitecture with timing speculation such as Paceline, cores arranged as pairs (usually two cores at close physical proximity) run the exact same thread [9]. One of the cores (*Leader*) is overclocked to the point of suffering occasional timing errors. The other core (*Checker*) runs at a safe clock frequency. The leader remains ahead of the checker, and by bringing data into the L2 cache that it shares with the checker, it prefetches for the checker — although its updates are prevented from reaching the L2 cache. Moreover, it also passes branch outcomes to the checker through a link called the Branch Queue (BQ), giving the checker near-perfect branch prediction. As a result, the checker keeps up with the accelerated leader. The net effect is that the thread executes faster than on a single base core. This idea is a variant of Slipstream [30], where the leader was faster (and incorrect) because it skipped some instructions.

To ensure correctness, there is hardware that periodically compares the architectural states in the two cores and recovers on a mismatch. Specifically, each core has logic that takes periodic register checkpoints and generates a hashed signature for each checkpoint. Then, there is a shared Validation Queue (VQ) that collects the signatures from the two cores and compares them. If there is a mismatch, a register checkpoint from the checker is copied to the leader.

As it executes, the overclocked leader dissipates higher power than a base core. Meanwhile, the checker leverages the branch outcomes and prefetches from the leader, executing far fewer wrong-path instructions and suffering fewer L2 misses. Together, the leader–checker pair dissipates only slighly more total power than two base cores. To manage the thermal imbalance that results from overclocking the leader, the two cores periodically alternate the leader position. Note that although the leader and checker cores redundantly execute the same thread, they do not execute in lock-step. Moreover, the cores can sever their Branch and Validation Queue links at any time and run different threads in *uncoupled* mode.

Figure 1(a) shows a typical curve of error rate (labeled P_E for probability of error) versus f for a core, where point A denotes the nominal operating point. As the core f increases from its nominal value f_0, first the guardband is

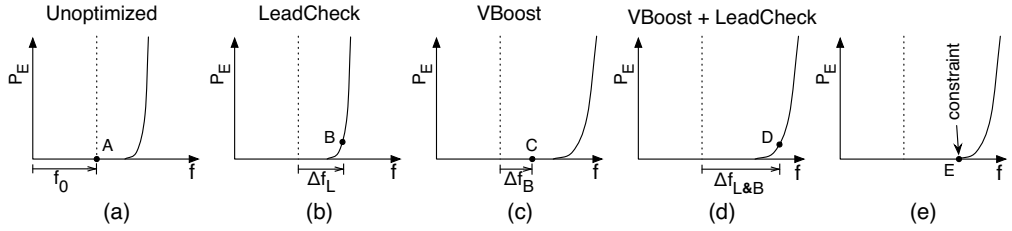


Figure 1: Qualitative depiction of increases in processor *f* when the *VBoost* and *LeadCheck* techniques are applied.

consumed and there are no errors. Once the guardband is exhausted, P_E starts increasing, first slowly, then rapidly. In these architectures, the optimal performance point is found when the leader runs at a *f* with a small P_E (point *B* in Figure 1(b)). For higher *f*, recovery costs negate any performance gains.

3 The Two Techniques are Complementary & Synergistic

A key novel insight of this paper is that the two techniques described are complementary and synergistic, multiplying their speedups on single-thread performance.

3.1 Characterizing Power Consumption

We call the techniques described in Sections 2.1 and 2.2 *VBoost* and *LeadCheck*, respectively. Of the two techniques, *VBoost* is simpler to implement. While *LeadCheck* is harder to build than *VBoost*, it entails fewer changes to the core than other major microarchitectural techniques for single-thread performance — such as thread-level speculation, core fusion/composition, or very wide superscalars. *LeadCheck* requires only the addition of a branch queue and validation/checkpointing logic. These are concentrated in the fetch and retire stages, where they are unlikely to affect critical paths.

Our goal is to apply these two techniques to threads that demand single-thread performance until the cores reach their maximum allowed power (P_{max}) or temperature (T_{max}). These techniques have different power behaviors. Per-core power (and, therefore, temperature) increases faster with *VBoost* than with *LeadCheck*. This arises from the fact that *VBoost* increases both V_{dd} and *f*, while *LeadCheck* only increases *f*. We can see this from the formulae for static power due to leakage (P_{leak} in Equation 6), dynamic power (P_{dyn} in Equation 5), and logic gate delay [23] (T_g in Equation 3).

$$V_t = V_{t0} + K_{temp}\,(T - T_0) + K_{DIBL}\,(V_{dd} - V_{dd0}) \quad (1)$$

$$\mu \propto T^{-1.5} \quad (2) \qquad T_g \propto \frac{V_{dd}\,L_{eff}}{\mu\,(V_{dd} - V_t)^{\alpha}} \quad (3)$$

$$I_{leak} \propto \frac{\mu}{L_{eff}}\,T^2\,e^{-q\,V_t/(k\,T\,n)} \quad (4)$$

$$P_{dyn} \propto C\,V_{dd}^2\,f \quad (5) \qquad P_{leak} = V_{dd}\,I_{leak} \quad (6)$$

As *VBoost* increases V_{dd}, both P_{dyn} and P_{leak} increase rapidly. Indeed, P_{leak} depends directly on V_{dd} (Equation 6) and, through I_{leak} (Equation 4), exponentially on V_{dd} (since V_t is a function of V_{dd} in Equation 1). Meanwhile, P_{dyn} has a quadratic dependence on V_{dd} (Equation 5) and a linear dependence on *f* (which itself depends on V_{dd}).

In general, when speeding up a single thread with either of these techniques, *VBoost* will reach power or temperature constraints at lower frequencies than *LeadCheck* — assuming that no other constraint limits the frequency increase earlier. One way to improve the effectiveness of *VBoost* is to add activity migration [11], which moves the thread among two available cores to reduce the formation of hotspots. We call the combination of *VBoost* and migration *VBo+Mig*.

3.2 Composing the Techniques

The two techniques are complementary because, in principle, the application of each one is limited by a different constraint. Specifically, suppose that we have as much power and temperature headroom as we want and use one of the techniques to gradually increase the frequency of a core. If we apply *LeadCheck*, the limiting factor will be the value of the error rate P_{Emax} beyond which the performance begins to drop because of the frequency of error recovery. Instead, if we apply *VBoost*, the limiting factor will be the value of the process maximum supply voltage (V_{ddmax}), beyond which the devices become unreliable.

Pictorially, this is shown in Figure 1. Figure 1(a) shows a typical P_E versus *f* curve for a core. The core works at *f*=*A*. If we apply *LeadCheck*, we eliminate the guardband and even allow the processor to tolerate some errors. We work at *f*=*B* in Figure 1(b), and accomplish an *f* increase of Δf_L. Instead, if we apply *VBoost* alone, the entire P_E versus *f* curve shifts to the right — while changing its slope in some way. Figure 1(c) shows that the processor can now work at *f*=*C* while still respecting the guardband. We have obtained an *f* increase of Δf_B. Finally, if we combine both *VBoost* and *LeadCheck*, we still have the same curve as Figure 1(c) but, as shown in Figure 1(d), we eliminate the guardband and tolerate some timing errors to operate at *f*=*D* for an *f* increase of $\Delta f_{L\&B}$.

In practice, the core may reach its power or temperature limits before the two techniques are applied to their full ex-

Regime	Bounding Constraints						Gain from Combining?	Multicore Loading Condition
	LeadCheck alone		*VBoost* alone		*VBo+LC*			
	T/P	P_E	T/P	V_{dd}	T/P	V_{dd}/P_E		
Individual	✓		✓		✓		Very Unlikely	Very High
	✓			✓	✓			
Synergistic		✓	✓		✓		Likely	High to Moderate
		✓		✓	✓			
Unfulfilled		✓		✓		✓	Definitely	Low

Table 1: Multicore constraint regimes. *T/P* stands for temperature or power.

tent. Figure 1(e) shows such a case where the *f* reaches only *E*. Note that the core running the critical thread reaches the temperature or power constraint sooner or later depending on the load in the rest of the multicore. Specifically, if the load is high, the temperature of the chip and, therefore, of the core is higher to start with. Moreover, a higher temperature induces higher leakage power, reducing the room to P_{max}. Overall, we define three operational *regimes*, as shown in Table 1.

In the *Individual* regime, the application of *LeadCheck* alone is sufficient to bring the core to its temperature or power limit (*T/P* in the table). The combination of the two techniques — which we refer to as *VBo+LC* — should not deliver further gains. This is because the voltage of the *LeadCheck* cores cannot be increased now without violating *T/P* constraints (as shown in the first row for *Individual* of Table 1). Alternatively, if we started with a *VBoost* configuration bounded by V_{dd} and applied *LeadCheck*, we could not get to a frequency higher than that of *LeadCheck* alone without violating *T/P* constraints (second row for *Individual* of Table 1). In general, this regime has a higher chance to occur when the rest of the chip is heavily loaded, since the other active cores generate a background of high temperature that reduces the room to T_{max} and P_{max} for the performance-critical core(s).

In the *Synergistic* regime, *LeadCheck* alone does not bring the core to its *T/P* limits. Instead, it reaches the maximum timing error rate first (P_E in the table). However, the application of both techniques together does bring the core to its *T/P* limits. In this regime, the combination of techniques is likely to deliver gains. In Section 6 we show that this regime dominates when the chip is under high to moderate load.

Finally, in the *Unfulfilled* regime, even when both techniques are applied together, the processor reaches the maximum supply voltage (V_{dd}) and limiting timing error rate (P_E) before arriving at the *T/P* constraint. In this case, the combination of techniques delivers gains, but there is power available for per-thread performance improvement that goes unused and could be spent on additional techniques if they were available. In Section 6 we demonstrate that this regime is common in lightly-loaded multicores.

3.3 A Configurable Multicore

To take advantage of these techniques, we propose the configurable multicore of Figure 2. To support *LeadCheck*, cores are grouped in pairs, where two cores share an L2 cache. The two cores also have a communication link for passing branch predictions from leader to the checker, and for comparing the hashed register checkpoints. When throughput matters, the two cores are decoupled; when *LeadCheck* is used to enhance single-thread performance, the cores are coupled and run the same thread, periodically swapping leader and checker roles.

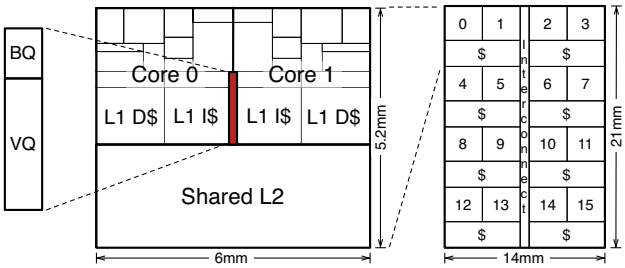


Figure 2: Core (left) and die (right) floorplan of the configurable multicore in 32-nm technology.

To support *VBoost* and *VBo+Mig*, each processor has its own *V* and *f* domain. When single-thread performance matters, the core's *V* and *f* are ramped up. Under *VBo+Mig*, the thread bounces between the core and its neighbor to equalize *T*. In this case, each of the two cores oscillates between a high-*V* and *f* mode and a shut-down mode. While we could migrate the thread to a remote core, the benefits of a shared cache more than compensate for the slightly higher *T* resulting from lateral heat conduction between the adjacent cores.

The configurable multicore operates as follows. At any time, there are *R* threads where throughput matters and *S* threads where single-thread performance is paramount. The multicore can assign each of the *S* threads either to a single core using *VBoost*, or to a core pair using *VBo+Mig*, *LeadCheck*, or *VBo+LC* (which is *VBoost* plus *LeadCheck*) — the decision is made by an optimizing controller. The *R* threads run normally on remaining cores, and any unused cores are shut down. As the load (*R*, *S*) changes, the controller reconfigures the mode, *f*, and V_{dd} of each core to maximize overall performance.

4 Dynamic Controller Design

To manage the configurability of the multicore, we design a hierarchical dynamic controller that runs in software. In this section, we formulate the problem, present the per-thread controllers, and finally present the global controller.

4.1 Problem Formulation

Our goal is to maximize the speed of the *S* threads, which are performance-critical. We do this by consuming all the power and thermal headroom available in the cores that run them. To ensure reliable operation, we must respect two constraints: the per-core power consumption must be less than P_{max}, and the hottest point in the core must have a temperature less than T_{max}. Additionally, to simplify the control, when using the *LeadCheck* technique, the timing error rate P_E is constrained to be less than $P_{Emax} = 10^{-5}$. It can be shown that this number is very close to the break-even point where the extra performance and time to recover from errors cancel each other. Note that we use per-core constraints rather than chip-wide ones because our hardware has independent per-core power grids and supplies, with no sharing of the current load between cores.

In this environment, we propose a thread controller in software per *S* thread (which may control one or two cores, depending on the technique used), and a simple global controller also in software that oversees the thread controllers. The inputs and outputs of each thread controller are:

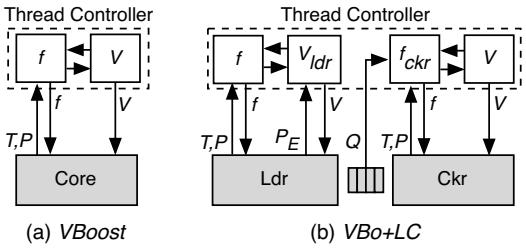
Inputs: The controller reads a power and a temperature sensor in the core (or each of the two cores) that it controls. This is like the Intel Foxton [20] controller. Additionally, if it controls two cores running in leader-checker mode, it reads two other sensors. The first one is a counter with the rate of checkpoint rollbacks observed — which gives the error rate P_E. The second one is the average occupancy of the leader-checker coupling queues — which gives an indication of whether the checker is falling behind the leader.

Outputs: The controller sets the voltage and frequency of the core (or each of the two cores) that it controls. We assume that *V* and *f* can be modified arbitrarily at every control interval (1 ms) with a minimum granularity of 10 mV and 100 MHz, respectively. The controller also sets the microarchitectural configuration in the cores (*VBoost*, *VBo+Mig*, *LeadCheck*, *VBo+LC*, or none).

4.2 Thread Controllers

A thread controller is implemented with two software modules: the *f*-subcontroller that sets the core frequency and the *V*-subcontroller that sets the core voltage. Next, we describe the way they work for each of the techniques.

Thread Controller for *VBoost*: In this technique, the thread controller increases the core's *f* as much as possible while applying an elevated *V*. Figure 3(a) shows the organization. The *f*-subcontroller drives the optimization by executing a simple search for the maximum *f* as it monitors the core's *P* and *T* for constraint violations. In the absence of violations, it attempts to increase *f* at the rate of 100 MHz every 2 ms. Otherwise, it initiates exponential back-off starting with a reduction of 100 MHz for the next control step.



(a) *VBoost* (b) *VBo+LC*

Figure 3: Thread controller examples.

Whenever the *f*-subcontroller generates a *f* change, it sends the new *f* to the *V*-subcontroller, which determines whether a feasible *V* exists to support it. If so, the *V*-subcontroller acknowledges the new *f* and instructs the core PLLs and supplies to move to the new operating point. Otherwise, it notifies the *f*-subcontroller that its *f* choice was infeasible and the *f*-subcontroller responds by proposing successively lower *f*. The *V*-subcontroller implementation is simple — just a lookup table indexed by *f*. At manufacturing test time, the table is populated with *V*s that guarantee error-free (fully guardbanded) operation of the core for different *f* settings.

Thread Controller for *VBo+LC*: In this technique, the thread controller drives the leader *f* as high as possible while keeping the error rate below P_{Emax}. Simultaneously, the checker frequency is adjusted to ensure that the checker does not fall behind. Both the leader and checker can receive elevated voltages. At all times, the checker is operated in a fully-guardbanded mode to ensure error-free operation just like a *VBoost*ed core.

As shown in Figure 3(b), this thread controller uses a *f*-subcontroller and a *V*-subcontroller for each of the two cores. Compared to *VBoost*, the leader's *V*-subcontroller (V_{ldr}) and the checker's *f*-subcontroller (f_{ckr}) are slightly different. The reason is that the V_{ldr}-subcontroller provides a *V* that accommodates a certain error rate, and the f_{ckr}-subcontroller has the goal of enabling the checker to catch up with the leader.

The controller works as follows. The leader uses the same *f*-subcontroller as in *VBoost* to continuously search for the highest *f*. However, the leader's V_{ldr}-subcontroller provides a *V* that accommodates the *f* target with an acceptable error rate P_E. Like in *VBoost*, the V_{ldr}-subcontroller is implemented as a lookup table indexed by *f*, but unlike in *VBoost*, it learns its entries dynamically in the field — since they additionally depend on application and environmental parameters. At power-on, all entries default to a minimum V_{dd}. Whenever a frequency f_{viol} is requested and P_{Emax} is exceeded, all table entries for $f \geq f_{viol}$ increment their

Chip parameters	
# cores: 16	Per-core max power P_{max}: 12W
Technology: 32nm	Heatsink thermal resistance R_{th}: $0.33K/W$
Core parameters	
Area: $3 \times 3mm$ (excluding L2)	Nominal unoptimized frequency: 5GHz
Width: 6-fetch 4-issue 4-retire OoO	L1 D Cache: 16KB WT, 2 cyc access, 4 way, 64B line
ROB: 152-entries	L1 I Cache: 16KB, 2 cyc access, 2 way, 64B line
Issue Window: 40 fp, 80 int	L2 Cache: 2MB WB, 2ns access, 8 way, 64B line,
LSQ Size: 54 LD, 46 ST	shared by two cores, has stride prefetcher
Branch pred: 80Kb tournament	Memory: 80ns round trip, 10GB/s max. per core
***LeadCheck* parameters**	
Max. error rate: P_{Emax}: 10^{-5} err/cyc	Max Leader–Checker lag: 512 instrs or 64 stores
Migration interval: $250\mu s$	

Table 2: Microarchitecture parameters.

V_{dd}	$0.8 - 1.3V$
V_{t0_nom}	$250mV$
T_0	85°C
T_{max}	100°C
K_{DIBL}	$-150mV/V$
K_{temp}	$-1.5mV/K$
n	1.5
α	1.3

Table 3: Technology parameters.

V_{t0} random σ/μ	6.4%
V_{t0} systematic σ/μ	6.4%
L_{eff} random σ/μ	3.2%
L_{eff} systematic σ/μ	3.2%
Corr. range (ϕ)	$1cm$

Table 4: Variation parameters.

V by $10mV$. In this way, the table entries monotonically converge toward the minimal *V*s that ensure $P_E < P_{Emax}$. However, because of the monotonicity, transient bursts of errors can drive *V* to values that are unnecessarily high in steady state. To combat this tendency, all entries are decremented by 10 mV once per second.

The f_{ckr}-subcontroller in the checker, rather than maximizing the checker's *f*, attempts to keep the leader–checker coupling queues no more than half and no less than one quarter full. It does so by treating the leader–checker pair as a GALS system and applying the *attack-decay* control algorithm [26] as follows. Whenever the queues are more than half full, the checker *f* is increased (*attack*) by 200 MHz per ms. When they are less than one quarter full, the *f* is decreased (*negative attack*) by 100 MHz per ms. Otherwise, the *f* is decreased by 100 MHz every 10 ms (*decay*).

Thread Controllers for *VBo+Mig* and *LeadCheck*: These controllers are special cases of the two organizations already presented. In *VBo+Mig*, each of the two cores has the *f*-subcontroller and *V*-subcontroller of *VBoost*. The only difference is that, in the inactive core, the *V*-subcontroller turns off the power supply to conserve power. The thread controller for *LeadCheck* is a degenerate case of *VBo+LC*'s where the two *V*-subcontrollers veto any *f* that is infeasible at the nominal *V*.

4.3 Global Controller

The global controller receives information from a higher level of the system (e.g., the operating system or the runtime) about which threads are *S* threads and which cores are idle. It then attempts to speed-up all the *S* threads and shut down the remaining idle cores. To keep the control simple, the global controller simply selects one technique, which all the thread controllers apply to the *S* threads — *VBoost*, *VBo+Mig*, *LeadCheck*, *VBo+LC*, or none. All thread controllers are asked to apply the same technique.

The algorithm used by the global controller is based on a greedy heuristic that works well in our experiments of Section 6. Specifically, if the chip has enough idle cores to dedicate two cores to each *S* thread, then the global controller selects *VBo+LC* for application. Otherwise, it selects *VBoost*. This heuristic is shown in Section 6 to be suboptimal only in rare cases when the chip is in the *Individual* regime of Section 3.2. Finally, the global controller reruns its algorithm every time when the number of *S* or *R* threads changes.

5 Experimental Setup

This section describes the modeling and optimization procedure used to evaluate LeadOut.

5.1 Microarchitecture Model

We model a cache-coherent multicore comprising 16 high-performance out-of-order cores, each configured as shown in Table 2. The core design is slightly more aggressive than current designs like the Intel Core i7 [13]. The cores contain the coupling link described in Section 3.3. Pairs of cores share an L2 cache. Each core has an independent V_{dd} and *f* domain, while the L2 caches have a fixed *f* and a fixed V_{dd} of 1V.

Thermal parameters are modeled on a desktop processor with a typical per-core power consumption of $6W$ and a maximum per-core power consumption P_{max} of $12W$ using a standard cooling solution. The floorplan, shown in Figure 2, is loosely based on the default EV6 floorplan from HotSpot [27].

5.2 Technology Model

We model a 32nm technology that is an incremental improvement on Intel's current high-performance 45nm process [3]. ITRS [15] projects that neither threshold nor supply voltage are scaling in the near term, so we use most values directly from [3] in our model, as shown in Table 3. The constant of proportionality for leakage in Equation 4 is set so that when all cores are active and under uniformly nominal conditions ($V_{dd} = 1V$, $T = 85°C$, and no process variation), the die expends 20W of leakage power (out of a total power of about 96W).

The maximum V_{dd} allowed by the process is an especially critical parameter that limits the performance scaling

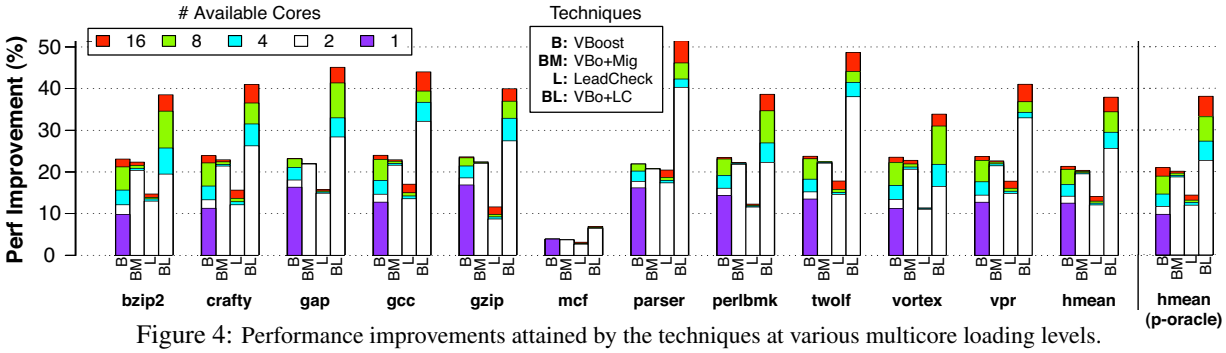


Figure 4: Performance improvements attained by the techniques at various multicore loading levels.

of *VBoost*. In the current 45nm process, the maximum V_{dd} ranges from 1.26V to 1.38V depending on the processor's current draw [13] (lower values at higher currents). These values are likely to decrease slightly in future technologies in order to meet reliability requirements. Therefore, for simplicity, we assume a maximum V_{dd} of 1.3V and a nominal value of 1V.

The frequency guardband is critical to the performance of *LeadCheck* schemes. We assume that during manufacturing test, the critical path delay of each core is measured at T_{max}. A worst-case timing margin – the guardband – is then added to this delay to obtain the rated clock period. Unfortunately, the exact guardbands for current processes are not publicly disclosed. For most of the evaluation, we set the guardband at 15%. Subsequently, Section 6.4.2 undertakes a sensitivity study.

To model process variation, we use VARIUS [24], with the parameters shown in Table 4. We model both random (spatially uncorrelated) and systematic (spatially correlated) variations in V_t and L_{eff}. The magnitude of the random V_{t0} variation assumed agrees with recent measurements [1].

5.3 Simulation Method

To measure performance and power consumption, we use a cycle-accurate timing simulator based on SESC [22], augmented with Wattch [4] in a closed loop with the leakage model of Section 3.1 and HotSpot [27] for temperature modeling. Die variation maps generated by VARIUS [24] inform the leakage calculation. VARIUS also calculates the per-core error rate for the current V_{dd}, T, and f conditions. We run all the SPECint2000 applications except for `eon`.

Given this optimization infrastructure, we perform three types of experiments. First, we characterize the performance and power of each technique. These experiments incorporate the effects of process variation. Specifically, each result is the average of 50 simulations, each using a different sample die variation map. Secondly, we construct a pseudo-oracle against which to compare the results of our proposed dynamic controller. The pseudo-oracle uses Nelder-Mead optimization [21] to generate static (i.e., not time-varying) values for the controller outputs from Section 4.1. Finally, we conduct sensitivity studies. For these experiments, we use only one typical die selected from among the 50 samples created before due to limitations in simulation time.

6 Evaluation

Our evaluation first examines the impact of each of the techniques for single-thread performance (Sections 6.1 and 6.2). In the process, it compares the performance of our thread controller to the pseudo-oracle, and identifies the key performance-limiting constraints under different loading conditions. Next, it characterizes the power-performance tradeoffs available with the different techniques (Section 6.3). It also provides a sensitivity study (Section 6.4). Finally, it charts the performance of Lead-Out on a workload requiring multiple high-speed threads (Section 6.5).

6.1 Performance

The performance of both *LeadCheck* and *VBoost* is determined by the application's responsiveness to frequency increase (i.e., the degree to which the application is CPU-bound) and the amount of frequency increase that is possible within the P_{max}, T_{max}, V_{ddmax}, and P_{Emax} constraints. We begin with the goal of speeding up a single target thread. This speedup will be highly dependent on the chip's loading conditions, which we measure in terms of the number of idle cores that are available, constituting the budget to run the target thread. An un-loaded chip would have all 16 cores available, while a maximally-loaded chip would have only one core available and the other cores would be active running other load. Note that *LeadCheck*, *VBo+Mig*, and *VBo+LC* require a core pair to run, so are only applicable when there are at least two cores available.

Figure 4 shows the performance improvement attained by the SPECint applications after applying each of the techniques. Recall that each result is the mean across all sample variation-afflicted dies. The improvements are over the application running on a plain core — an environment we call *Unoptimized*. For each application, the individual bars correspond to the different techniques. In a given bar, we stack the improvements attained under different amounts of load in the multicore — measured in number of available cores. Specifically, the top segment represents the lightest

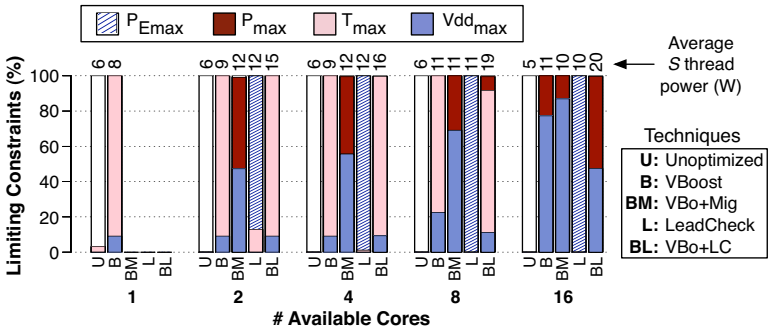


Figure 5: Relative incidence of limiting constraints across all applications and die samples. The numbers above the bars show the average power consumption of the *S* thread on each system (summing the contributions of two cores, if applicable).

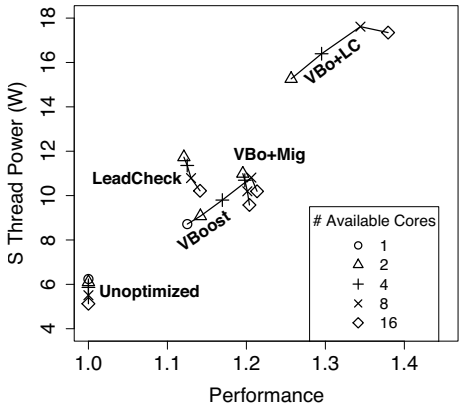


Figure 6: Per-thread power consumption vs. performance under different load conditions.

possible load, where all 16 cores are available to run the critical thread; the bottom segment represents the heaviest load, where 15 cores are busy and only one core is available. The figure also shows the harmonic mean (*hmean*) and the harmonic mean with oracle (*hmean p-oracle*). We will discuss the latter later.

In situations with just one core available, the only applicable technique is *VBoost*. Looking at *hmean*, *VBoost* improves performance by 12% on average. If two cores are available, *LeadCheck* comes online to provide a performance gain of 12% in *hmean*, compared to *VBoost*'s 14%. Note that even at such high loading, the system is at least in the *Synergistic* regime. This is seen by combining *LeadCheck* and *VBoost* into *VBo+LC*, which delivers a 26% average performance improvement. The *Individual* regime is not seen in these experiments. While certain dies may experience the *Individual* regime, the average of the 50 runs never does. This can be seen from the fact that *VBo+LC* always outperforms *LeadCheck*.

For all loads, *VBo+Mig* in *hmean* provides a consistent and compelling improvement of 20% over *Unoptimized*. In some lightly-loaded systems, it is slightly outperformed by *VBoost*. This is due to an artifact of our thread controller implementation[1]. Finally, *VBo+LC* performs much better than either technique alone — especially as the number of available cores increases. When all 16 cores are available, we can see from *hmean* that *VBo+LC*'s average performance improvement is 38%.

The rightmost group of bars (*p-oracle*) shows the performance of the pseudo-oracular optimizer of Section 5.3. We see that the real controller performs comparably to the pseudo-oracle — although the latter does not provide a strict upper bound on performance because its outputs are static.

[1]For simplicity, we set the *f* of both cores in the *VBo+Mig* pair to be that of the slower core in the pair. This makes it slightly slower than *VBoost* under best-case conditions.

6.2 Limiting Constraints for Each Technique

To explain these performance trends, Figure 5 shows the limiting constraints for each technique and loading level. The size of each bar segment represents the fraction of samples (out of 550, given by 50 dies times 11 applications) from the pseudo-oracular experiments in which the particular constraint was the *limiting* one. The constraints can be P_{Emax}, P_{max}, T_{max}, or V_{ddmax}. The number above each bar is the average power consumption of a latency-critical (S) thread under that configuration (adding the contributions of two cores, if applicable).

The three regimes of Table 1 appear in the figure as follows. When *LeadCheck* is limited by P_{max} or T_{max}, the system is in the *Individual* regime. This regime is seen in about 10% of the cases when two cores are available. When *LeadCheck* is limited by P_{Emax} and *VBo+LC* is mostly limited by P_{max} or T_{max}, the system is in the *Synergistic* regime. This is the dominant regime for 2-8 available cores. It also appears 50% of the time for 16 available cores. Finally, when *VBo+LC* is often limited by P_{Emax} or V_{ddmax}, it is *Unfulfilled*. This is seen 50% of the time for 16 available cores.

In a heavily-loaded system (e.g., one core available), the high total power consumption of all the cores on chip raises the heatsink temperature, reducing thermal headroom. Consequently, when we apply *VBoost*, it increases the voltage only slightly before hitting the temperature limit. The result is that *VBoost* provides only modest performance improvements on a fully-loaded system.

When the system has two available cores, *VBoost* remains temperature limited, able to dissipate only 9W on average. *VBo+Mig* addresses the thermal problem effectively, allowing the thread to dissipate 12W (over two cores) on average and reach the supply voltage limit in many cases. Referring to Figure 4, this provides substantial marginal performance gains over *VBoost*. Meanwhile, *LeadCheck* opens a new avenue for performance enhancement: *LeadCheck* is limited by P_{Emax} and has ample room to

grow in power and temperature. In other words, *LeadCheck* provides an *orthogonal* means of increasing performance. When *VBoost* and *LeadCheck* are used together in *VBo+LC*, they work in the *Synergistic* regime. Referring to Figure 4, they attain substantial improvements. They are bounded mainly by temperature.

The cases of four to eight available cores show that *LeadCheck* is usually limited by P_{Emax} and *VBo+Mig* is often limited by V_{ddmax}, but by applying *VBo+LC*, we usually arrive at the power or temperature limits. This is still the *Synergistic* regime.

In unloaded systems (16 available cores), *VBoost* and *VBo+Mig* are practically limited by V_{ddmax}, while *LeadCheck* is limited by P_{Emax}. After combining them in *VBo+LC*, the system is still limited by V_{ddmax} about 50% of the time. In these cases, the system works in the *Unfulfilled* regime. It has left-over power and temperature headroom to accommodate other techniques that increase single-thread performance.

Overall, the key benefit of combining *LeadCheck* with *VBoost* is that they provide orthogonal means of improving performance because they have different limiting constraints. This leads to the *Synergistic* and *Unfulfilled* regimes.

6.3 Power–Performance Tradeoffs

To fully assess the different techniques, Figure 6 shows the performance and power consumption of a latency-critical (*S*) thread under the different techniques and a range of loading conditions. The power and performance reported are the mean across all applications and dies. The performance is normalized to the *Unoptimized* environment. In the figure, each curve corresponds to one technique. The diamond point on each curve represents an unloaded system with 16 available cores, while the circle (or the triangle in the case of *LeadCheck*, *VBo+Mig*, and *VBo+LC*), represents a heavily-loaded system. Other markers on each curve identify the remaining loading points from Figures 4 and 5.

This plot shows the *composability* of the *LeadCheck* and *VBoost* (or *VBo+Mig*) techniques, and the great *capability* of the *VBo+LC* technique. Indeed, the figure shows that, roughly speaking, *LeadCheck*, *VBoost*, or *VBo+Mig* deliver at best a 20% improvement in single-thread performance at the cost of doubling the power of the thread. However, *VBo+LC* delivers a 38% improvement in single-thread performance at the cost of trebling the thread power. This is a remarkably high performance improvement. It is unreachable with either base technique alone, and shows that the two base techniques are orthogonal. For the arguably more likely case when only 8 cores are available, *VBo+LC* improves single-thread performance by 34% while consuming 220% more power than *Unoptimized*.

The figure also shows that, in practice, *VBoost* (and *VBo+Mig*) are more power-efficient than *LeadCheck*. At approximately the same power, *VBoost* and *VBo+Mig* deliver higher performance than *LeadCheck*.

Finally, we note that power consumption does not increase monotonically with performance. For most techniques, power decreases slightly for the highest performance point — namely, the one with the lowest load. This is due to the decrease in leakage power at the lower die temperatures that prevail under light load. When most of the cores on the die are idle, the total chip power dissipation is low, so the heatsink is cold. A low temperature means that leakage is low.

6.4 Sensitivity Analysis

We now characterize the sensitivity of the performance improvements. In many cases, T_{max} is a limiting factor, so we examine different thermal design points. Additionally, *LeadCheck* systems achieve their performance improvements primarily by consuming guardband, so we evaluate several different guardband sizes. Finally, we consider different per-core power limits that correspond to weaker and more robust per-core power grids. Figure 7 shows the three sensitivity studies, varying the parameters that have been assumed for the preceding experiments (labeled as "default"). All numbers are normalized to the performance of *Unoptimized* with the same parameters under the same conditions. Note that the default-parameter bars are not exactly like those in Figure 4 because here, we only simulate a single (typical) die, rather than 50 samples. In the following, we consider each plot in turn.

6.4.1 Power/Thermal Envelope

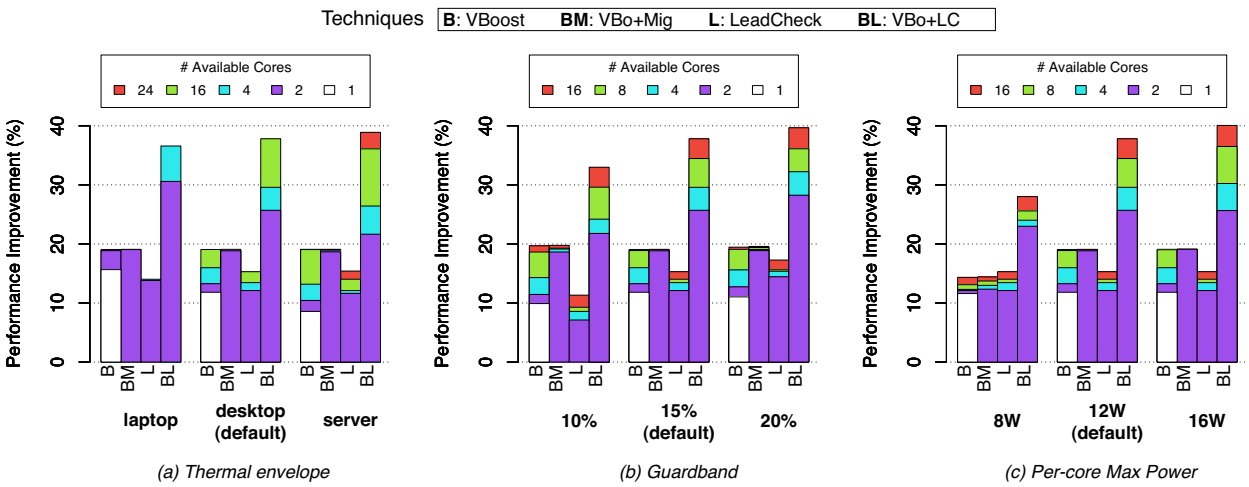
We consider three different power/thermal environments, namely *laptop*, *desktop*, and *server*, as shown in Table 5. The configurations vary the total number of cores on the chip to fit into their respective thermal envelopes — 4 cores in *laptop*, 16 cores in *desktop*, and 24 cores in *server*. Additionally, *server* assumes a powerful high-airflow cooling system typical of datacenter installations, while *laptop* is constrained by a relatively weak heatsink due to its physical confines[2].

	Laptop	Desktop	Server
# Cores	4	16	24
TDP (W)	24	96	144
Heatsink R_{th} (K/W)	1.3	0.33	0.30
Ambient T (K)	313	313	305

Table 5: Power/thermal environments for the sensitivity study.

Figure 7(a) shows that the performance trends are very similar in all three thermal environments. For example, in an unloaded system, the performance gains for each technique are very close across all environments. This is a result of changing the number of cores in the chip at the same

[2] *Laptop*, *desktop*, and *server* heatsinks are modeled on Aavid Thermalloy types 3680, 1002/D, and 037704, respectively.



(a) Thermal envelope *(b) Guardband* *(c) Per-core Max Power*

Figure 7: Sensitivity of performance improvements to the thermal environment (a), process guardband (b), and power grid design (c).

time as we change the quality of the heatsinks.

6.4.2 Guardband

Figure 7(b) shows that, as the process guardband increases from 10% to 20%, the effectiveness of *LeadCheck* also grows. This is because *LeadCheck* gets much of its speedup from removing the guardband (refer to Figure 1(b)). As the guardband increases from 10% to 15%, the returns from *LeadCheck* grow by roughly 0.04. However, if the guardband increases above 15%, the performance benefits from *LeadCheck* rise at a slower rate — tempered by the responsiveness of the application performance to frequency increases and by the lower frequency of the checker core.

6.4.3 Power Grid Capacity

Figure 7(c) shows how increasing the per-core power constraint P_{max} impacts performance. From a design perspective, supporting a higher P_{max} requires a more robust per-core supply grid with lower resistance, higher decoupling capacitance, and lower inductance, which can be difficult to achieve. The *default* design assumed a grid capable of supplying a worst-case power equal to twice the average power consumption of the applications, which we believe represents a typical design. Decreasing that budget by one third to arrive at $P_{max} = 8W$ degrades all techniques except *LeadCheck*, but they still provide compelling speedups. Increasing the budget by one third to $16W$ benefits only the most power-hungry cases and even then, only slightly. The heavily-loaded systems benefit little because they are temperature-limited. Overall, we conclude that none of the techniques requires more than a standard power distribution system.

6.5 Configurability

The previous evaluations focused on the goal of optimizing *a single thread*, but when executing a multiprogrammed workload or an application with some, limited parallelism, it will often be necessary to speed up a *larger number S* of latency-critical threads (Section 3.3). Here, the key question is how much performance can be delivered to the S latency-critical threads while the chip concurrently runs a specified number *R* of throughput-oriented threads.

In this section, we again use 16 cores per chip. Recall that *LeadCheck*, *VBo+Mig*, and *VBo+LC* all use two cores to speed-up a single S thread. Therefore, they can execute at most $S = 8$ latency-critical threads when the system is otherwise unloaded. Only *VBoost* is able to speed-up $S > 8$ threads.

Figure 8 shows the percentages of performance improvement experienced by an *S* thread for each technique when the application demands different numbers of R and S threads. Plots (a)-(d) correspond to one technique (or a fixed combination of them) each, while plot (e) corresponds to the algorithm used by the global controller (Section 4.3), namely choose *VBo+LC* if there are twice as many idle cores as *S* threads, or *VBoost* otherwise. Each plot shows iso-performance contours for the *S* threads in increments of 2%. Performance is normalized to that of an unoptimized chip running the same number of threads. For techniques that use two cores per *S* thread (plots (b), (c), and (d)), the region where $R + 2S > 16$ is infeasible; for *VBoost* (plot (a)), the region where $R + S > 16$ is infeasible. Finally, plot (e) is an "overlay" of plot (d) when *VBo+LC* is feasible and plot (a) otherwise.

For their feasible configurations, both *VBo+Mig* and *LeadCheck* offer consistent per-thread performance improvement, varying by less than 2% over the entire feasible range of *S* and *R*. When available, *VBo+LC* always offers superior performance improvements to either technique alone. However, that performance is more sensitive to the particular value of *S* and *R* demanded. Likewise, the performance of *VBoost* falls off as the number of *S* threads approaches the limit.

Plot (e) demonstrates the benefits of having a configurable multicore. When the chip must run many S and R

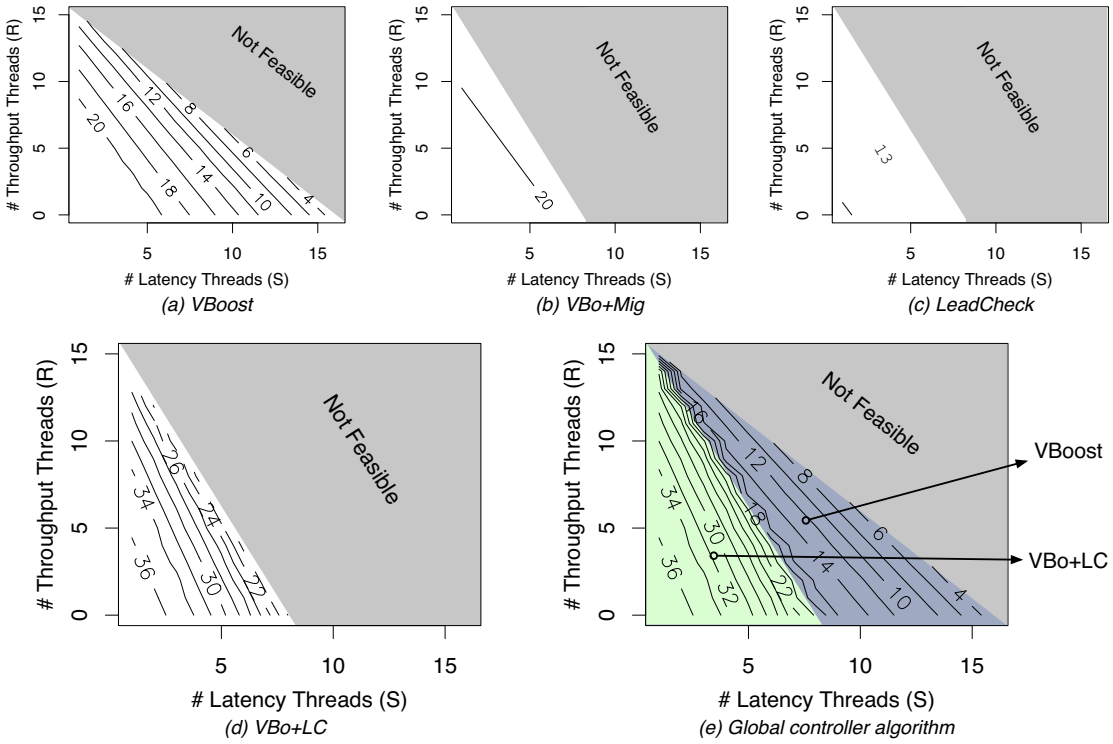


Figure 8: Performance improvements (in percent) experienced by each *S* thread when the application demands different numbers of *R* and *S* threads. For each technique, there is an area of infeasible configurations. Iso-performance contours are shown in the feasible regions.

threads at the same time (darker region), the system uses *VBoost* and attains good performance. When the number of *S* threads is low or the load is light (light region), it then switches to *VBo+LC*, delivering higher performance than *VBoost* alone would deliver. If the system only had *LeadCheck*, it would provide less performance increase in the light region and would be helpless in the dark region.

7 Related Work

Architects have long recognized the need for configurable multicores. Slipstream [12] used cores in coupled mode to execute parallel applications and found that, even for the highly-parallel SPLASH-2 benchmarks, a 32-processor system running 16 threads in coupled mode usually outperformed the same system running twice as many threads in uncoupled mode. More recently, Core Fusion [16] and TFlex [17] have shown speedups over static heterogeneous multicores on a wide range of applications by combining pipeline resources from neighboring cores to improve the IPC of critical threads.

Although it uses similar hardware support, the *VBoost* technique is distinct from DVFS, which has been proposed and used [5] to minimize power consumption by throttling back core frequency and voltage when performance is not required. *VBoost* applies the concept in reverse, to *increase* performance when possible. Intel's Turbo Boost [14] is the first commercial example.

Leader-checker schemes such as Slipstream [12, 30], Dual-Core Execution [31], Future Execution [8], and Paceline [9] are a somewhat new development emerging in an era of plentiful cores. They allow two cores to be coupled in executing a single thread, but each boosts the leader performance in different ways: Paceline relies on timing speculation, Slipstream elides instructions, Dual-Core Execution speculates past cache misses, and Future Execution uses value prediction.

The combination of timing speculation with DVFS has also been examined before. Specifically, Razor [7] combined voltage reduction with latch-level timing speculation and error correction to save power. Similarly, EVAL [25] proposed trading off performance, error rate, and power in a configurable core with timing speculation support. EVAL differs from LeadOut in three key ways. First, EVAL uses several intra-core supply voltage and frequency domains, while LeadOut uses only one domain per core. Second, EVAL's timing speculation is not configurable, since its per-core DIVA-style [2] checker is always on, consuming power and area. Finally, EVAL configures microarchitecture structures inside the core (e.g., the size of the issue queue or the body bias applied to the execution unit), while

LeadOut considers a core as a unit.

8 Conclusion

An intuitive approach to improve single-thread performance is to shut down the idle cores in the chip and run the busy, performance-critical ones at higher-than-nominal frequencies. To enable such frequencies, two relatively low-overhead techniques either boost voltage beyond nominal values, or pair cores in leader-checker configurations and let them run beyond safe margins.

This paper has observed that, in a large multicore with varying numbers of busy cores and critical threads, individual application of either of these two techniques alone is suboptimal. Due to supply voltage or error rate limitations, they are often unable to bring the multicore all the way to the extremes of its power or temperature envelopes. Moreover, this paper also showed that these two techniques are complementary and can be synergistically combined to unlock much higher levels of single-thread performance. Finally, the paper demonstrated a dynamic controller that optimizes the two techniques.

We evaluated these techniques and their combination on a simulated 16-core configurable multicore. We saw that, when half of the cores are busy and we want to run one additional, performance-critical thread, application of either leader-checker execution or voltage boosting increases the thread's performance by about 20% at most, while increasing its power by about 100%. However, if we combine both techniques, the thread attains 34% higher performance, while consuming 220% more power. Similar gains are enabled for various load conditions.

References

[1] K. Agarwal and S. Nassif. Characterizing process variation in nanometer CMOS. In *Design Automation Conference*, June 2007.

[2] T. Austin. DIVA: A reliable substrate for deep submicron microarchitecture design. In *International Symposium on Microarchitecture*, November 1999.

[3] C. Auth et al. 45nm high-k+metal gate strain-enhanced transistors. *Intel Technology Journal*, 12(2), June 2008.

[4] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *International Symposium on Computer Architecture*, June 2000.

[5] T. Burd and R. Broderson. Processor design for portable systems. *Journal of VLSI Signal Processing Systems*, 13(2), August 1996.

[6] J. Dorsey, S. Searles, M. Ciraula, S. Johnson, N. Bujanos, D. Wu, M. Braganza, S. Meyers, E. Fang, and R. Kumar. An integrated quad-core Opteron processor. In *International Solid State Circuits Conference*, February 2007.

[7] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Zeisler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. Razor: A low-power pipeline based on circuit-level timing speculation. In *International Symposium on Microarchitecture*, December 2003.

[8] I. Ganusov and M. Burtscher. Future execution: A prefetching mechanism that uses multiple cores to speed up single threads. In *ACM Transactions on Architecture and Code Optimization*, volume 3, 2006.

[9] B. Greskamp and J. Torrellas. Paceline: Improving single-thread performance in nanoscale CMPs through core overclocking. In *International Conference on Parallel Architectures and Compilation Techniques*, September 2007.

[10] L. Hammond, M. Willey, and K. Olukotun. Data speculation support for a chip multiprocessor. In *International Conference on Architectural Support for Compilers and Operating Systems*, October 1998.

[11] S. Heo, K. Barr, and K. Asanović. Reducing power density through activity migration. In *International Symposium on Low Power Electronics and Design*, August 2003.

[12] K. Ibrahim, T. Byrd, and E. Rotenberg. Slipstream execution mode for slipstream-based multiprocessors. In *International Symposium on High Performance Computer Architecture*, February 2003.

[13] Intel. Intel Core i7 processor Extreme Edition and Intel Core i7 processor datasheet, November 2008.

[14] Intel. Intel Turbo Boost technology in Intel Core microarchitecture (Nehalem) based processors, November 2008.

[15] International Technology Roadmap for Semiconductors. (2007 Edition), 2007.

[16] E. Îpek, M. Kırman, N. Kırman, and J. F. Martínez. Core fusion: Accommodating software diversity in chip multiprocessors. In *International Symposium on Computer Architecture*, May 2007.

[17] C. Kim, S. Sethumadhavan, M. Govindan, N. Ranganathan, D. Gulati, D. Burger, and S. Keckler. Composable lightweight processors. In *International Symposium on Microarchitecture*, December 2007.

[18] W. Kim, M. Gupta, G. Wei, and D. Brooks. System level analysis of fast, per-core DVFS using on-chip switching regulators. In *International Symposium on High Performance Computer Architecture*, February 2008.

[19] V. Krishnan and J. Torrellas. A chip-multiprocessor architecture with speculative multithreading. *IEEE Trans. on Computers*, September 1999.

[20] R. McGowen, C. A. Poirier, C. Bostak, J. Ignowski, M. Millican, W. H. Parks, and S. Naffziger. Power and temperature control on a 90-nm Itanium family processor. *IEEE Journal on Solid State Circuits*, 41(1), January 2006.

[21] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, (7), 1965.

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

[23] T. Sakurai and R. Newton. Alpha-power law MOSFET model and its applications to CMOS inverter delay and other formulas. *IEEE Journal on Solid State Circuits*, 25(2), 1990.

[24] S. Sarangi, B. Greskamp, R. Teodorescu, A. Tiwari, and J. Torrellas. VARIUS: A model of process variation and resulting timing errors for microarchitects. *IEEE Transactions on Semiconductor Manufacturing*, 21(1), February 2008.

[25] S. R. Sarangi, B. Greskamp, A. Tiwari, and J. Torrellas. EVAL: Utilizing processors with variation-induced timing errors. In *International Symposium on Microarchitecture*, November 2008.

[26] G. Semeraro, D. Albonesi, S. Dropsho, G. Magklis, S. Dwarkadas, and M. L. Scott. Dynamic frequency and voltage control for a multiple clock domain microarchitecture. In *International Symposium on Microarchitecture*, November 2002.

[27] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *International Symposium on Computer Architecture*, June 2003.

[28] G. Sohi, S. Breach, and T. Vijaykumar. Multiscalar processors. In *International Symposium on Computer Architecture*, June 1995.

[29] G. Steffan and T. Mowry. The potential for using thread-level data speculation to facilitate automatic parallelization. In *International Symposium on High Performance Computer Architecture*, February 1998.

[30] K. Sundaramoorthy, Z. Purser, and E. Rotenberg. Slipstream processors: Improving both performance and fault tolerance. In *International Symposium on Architectural Support for Programming Languages and Operating Systems*, November 2000.

[31] H. Zhou. Dual-core execution: Building a highly scalable single-thread instruction window. In *International Conference on Parallel Architectures and Compilation Techniques*, September 2005.