Computational Sprinting on a Hardware/Software Testbed

Arun Raghavan* Laurel Emurian* Lei Shao‡
Marios Papaefthymiou† Kevin P. Pipe†‡ Thomas F. Wenisch† Milo M. K. Martin*

* Dept. of Computer and Information Science, University of Pennsylvania
† Dept. of Electrical Engineering and Computer Science, University of Michigan
‡ Dept. of Mechanical Engineering, University of Michigan

Abstract

CMOS scaling trends have led to an inflection point where thermal constraints (especially in mobile devices that employ only passive cooling) preclude sustained operation of all transistors on a chip—a phenomenon called "dark silicon." Recent research proposed computational sprinting*—exceeding sustainable thermal limits for short intervals—to improve responsiveness in light of the bursty computation demands of many media-rich interactive mobile applications. Computational sprinting improves responsiveness by activating reserve cores (parallel sprinting) and/or boosting frequency/voltage (frequency sprinting) to power levels that far exceed the system's sustainable cooling capabilities, relying on thermal capacitance to buffer heat.*

Prior work analyzed the feasibility of sprinting through modeling and simulation. In this work, we investigate sprinting using a hardware/software testbed. First, we study unabridged sprints, wherein the computation completes before temperature becomes critical, demonstrating a 6.3× responsiveness gain, and a 6% energy efficiency improvement by racing to idle. We then analyze truncated sprints, wherein our software runtime system must intervene to prevent overheating by throttling parallelism and frequency before the computation is complete. To avoid oversubscription penalties (context switching inefficiencies after a truncated parallel sprint), we develop a sprint-aware task-based parallel runtime. We find that maximal-intensity sprinting is not always best, introduce the concept of sprint pacing*, and evaluate an adaptive policy for selecting sprint intensity. We report initial results using a phase change heat sink to extend maximum sprint duration. Finally, we demonstrate that a sprint-and-rest operating regime can actually outperform thermally-limited sustained execution.*

1. Introduction

The anticipated end of CMOS voltage (a.k.a. Dennard) scaling has led to an inflection point in computer system design. In the past, chip designers could deliver value by exploiting transistor counts that double with near-constant total chip power each technology generation. Technology trends indicate that in the future, although transistor dimensions will likely continue to scale for at least another decade, power density will grow with each generation at a rate that far outstrips improvements in our ability to dissipate heat [11, 15, 18, 20, 48, 52]. This conundrum has led many researchers and industry observers to predict the advent of "dark silicon", *i.e.*, that much of a chip must be powered off at any time [11, 12, 18, 20, 26, 37, 48, 52]. Thermal constraints are particularly acute in hand-held and mobile devices that are restricted to passive cooling; dark silicon is already a reality in mobile chips like the Apple A5, where half of chip area is dedicated to accelerators that are active only some of the time.

Whereas the thermal constraints that underlie dark silicon loom as a significant industry challenge, they create an opportunity to rethink how we use chip area to deliver value. Many interactive applications are characterized by short bursts of intense computation punctuated by long idle periods waiting for user input [8, 47, 54]. Media-intensive mobile applications, such as mobile visual search [2, 3, 23], handwriting and character recognition [17, 35], and augmented reality [53], often fit this pattern. For short bursts of computations found in such applications, one approach to improve responsiveness on thermally-constrained devices is to transiently exceed a chip's sustainable thermal limits to deliver a brief but intense burst of computation.

Our prior work proposed *computational sprinting* [45], which activates reserve cores (parallel sprinting) and/or boosts frequency and voltage (frequency sprinting) to power levels that exceed the system's sustained cooling capabilities (*i.e.*, the thermal design power or TDP) by an order of magnitude or more. Sprinting exploits *thermal capacitance*, the property that materials can buffer significant heat as they rise in temperature and subsequently dissipate this heat to the ambient after the sprint. Recent chips from Intel and AMD already exploit such phenomena with limited forms of frequency sprinting. For example, Intel's second-generation "Turbo Boost" exceeds the sustainable chip power by 25% for tens of seconds [46].

Previously, we explored the feasibility of computational sprints that exceed sustainable thermal limits by an order of magnitude through a combination of modeling and simulation [45]. In this paper, we describe and measure a concrete implementation of computational sprinting in a hardware/software testbed that supports both parallel and frequency sprinting, offering a maximum sprint intensity that exceeds sustainable power levels by a factor of five. The testbed comprises a conventional quad-core Core i7 desktop system, in which we remove the chip's heat sink to engineer a system with a 10 W TDP—just enough cooling to sustain the indefinite operation of a single core at the lowest configurable operating frequency. At the maximum sprint intensity, with four cores at maximum configurable frequency drawing over 50 W, chip temperature will rise to cause overheating in mere seconds. However, this temperature rise is not instantaneous, but extends over a few seconds because of the inherent thermal capacitance of the copper heat spreader and chip package; we exploit this brief interval to sprint. We implement a software runtime system to monitor and control the platform and execute a suite of image processing and vision workloads to investigate computational sprinting on this testbed.

We first study *unabridged sprints*—those sprints that can be completed within the thermal capacitance limits of our testbed. We show that the thermal capacitance of the heat spreader is sufficient to allow a few seconds of 5× sprints over the platform's sustainable thermal dissipation, resulting in an average improvement in

ASPLOS'13, March 16–20, 2013, Houston, Texas, USA.

Reprinted from ASPLOS'13,, [Unknown Proceedings], March 16–20, 2013, Houston, Texas, USA., pp. 1–12.

responsiveness of 6.3× over sustainable execution. We also analyze the energy impact of sprinting and find that even for a chip that has not been designed for sprinting, using parallel sprinting can result in a net energy efficiency gain relative to sustainable operation. This somewhat counterintuitive result stems from the relatively high background power required to activate even a single core (*e.g.*, last-level cache and other "uncore" logic), so the incremental performance benefit of activating additional cores more than offsets the additional energy consumption. Furthermore, by completing the computation more quickly, the system can enter idle mode more quickly. Our analysis indicates that the energy efficiency advantages of sprinting grow as idle power decreases. Hence, although there is already a more than ten-to-one ratio between peak and idle power in the testbed's chip, even lower idle power remains desirable in chips designed with sprinting in mind.

Overall, when sprints can complete within thermal capacitance limits, the sprinting policy is relatively straightforward on the testbed: for best responsiveness, sprint with all cores at maximum frequency and voltage; for best energy efficiency, sprint with all cores at the minimum operating frequency and voltage. However, experiments reveal that optimal sprinting policy becomes much more complicated when the thermal capacitance is insufficient to complete all work during a sprint. We call sprints that must end due to thermal limits *truncated sprints*.

We study two aspects of truncated sprints. First, we study the *oversubcription penalty* of post-sprint execution. When a parallel sprint is truncated, execution switches to reach a sustainable operating mode by disabling all but one core and migrating all still-active threads to be multiplexed on a single core. This coarse-grain multiplexing by the operating system can lead to substantial slowdown relative to simply running the entire computation on a single thread, particularly when threads synchronize frequently [9, 25, 28, 29, 50]. If the post-sprint execution period is large relative to the sprint, then sprinting can result in a net slowdown; both energy and responsiveness would have been better if the system had never attempted to sprint at all. This effect is most pronounced on one workload, where the oversubscription penalty is nearly 2×. We show that converting the workload to use a sprint-aware task-based work-stealing parallel runtime can serve as a general strategy to eliminate this oversubscription penalty. In this sprint-aware runtime, worker threads simply cease claiming new tasks and instead put themselves to sleep as thermal capacity nears exhaustion.

Addressing the oversubscription problem ensures that sprinting will not result in performance worse than sustainable single-core execution. Nevertheless, simply sprinting at maximum intensity is not always best. Rather, experiments show that maximum responsiveness requires *sprint pacing* in which a sprint intensity is selected such that the computation completes just as the thermal capacitance is exhausted. Optimum sprint pacing depends on the interplay of available thermal capacitance, the amount of work to be completed, and the power/performance characteristics of the platform and workloads. Measured results indicate that the best static sprint intensity can be determined *a posteriori* once the amount of work to complete is known. However, as the required length of a sprint is often uncertain, *a priori* sprint pacing (*e.g.*, based on predicting task lengths) may not be feasible. Instead, we propose a dynamic pacing scheme in which sprints are initiated at maximum intensity, but then frequency (and voltage) are reduced after half of the thermal capacitance has been consumed. Although straightforward, this policy provides the maximum responsiveness for short sprints while providing the benefits of sprinting to a wider range of computation lengths than maximum-intensity sprinting alone.

Prior work suggested incorporating phase change materials into the chip packaging to extend maximum sprint duration with the additional thermal capacitance available in the latent heat of a phase transition [45]. We report on experiences with augmenting the testbed with a phase change heat sink placed on top of the package, demonstrating a 6× extension in sprint duration.

Finally, we consider the implications of computational sprinting for long-running computations that greatly exceed the maximum sprint duration. Although intuition might suggest such computations call for sustained execution within the thermal limit, we find that a staccato *sprint-and-rest* operating regime is both faster and more energy efficient. Sprint-and-rest outperforms thermally-constrained sustained operation because energy efficiency is maximized by activating all useful cores, thus amortizing the fixed costs of operating at all.

To summarize, this paper is the first to explore aspects of sprinting on an actual hardware testbed (Section 2), and the paper's experimental findings include:

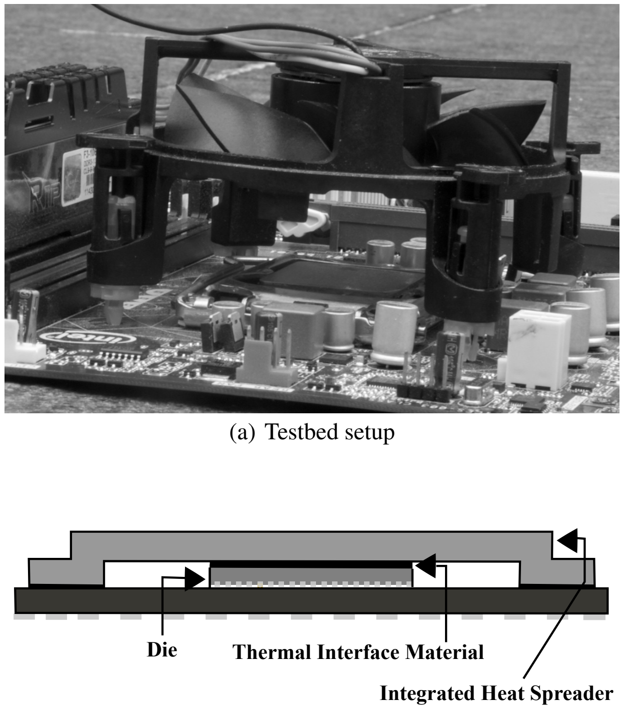
- Showing that existing sources of thermal capacitance are sufficient for 5× intensity sprints for up to a few seconds of duration (Section 3.1).
- Demonstrating that parallel sprinting can actually improve energy efficiency (Section 3.2).
- Identifying the potential inefficiencies of truncating sprints and proposing a sprint-aware task-based runtime as a remedy (Section 4.2).
- Introducing the concept and opportunity of *sprint pacing* and proposing an initial adaptive sprint pacing policy implemented in the software runtime (Section 4.3).
- Reporting our experiences with augmenting the testbed with a phase-change heat sink (Section 5).
- Demonstrating that sprint-and-rest outperforms thermally constrained (sustained) execution for long-running computations (Section 6).

2. Sprinting Testbed Hardware

In contrast to earlier simulation-based studies [45, 49], this paper investigates computational sprinting using a testbed built from a real system. We modify a current desktop system to act as a proxy for a hypothetical future thermally-limited mobile system. Avoiding simulation overcomes modeling artifacts and simulation time constraints [38], but it is imperfect in that our study is limited to existing chips/platforms, which have not been designed with sprinting in mind.

To evaluate sprinting, a chip must offer an operating point where its peak power greatly exceeds the sustainable power dissipation of its cooling system (*i.e.*, the platform's TDP). Existing mobile chips have been designed with peak power envelopes easily dissipated via passive cooling, and hence are not appropriate for our study. Instead, as a proxy for the thermal characteristics of a future sprint-enabled device, we create a sprinting testbed system using a multi-core desktop-class chip and adjust its cooling environment to create an appropriate ratio between the chip's peak power and the platform's sustainable TDP (in our case, 5:1). A further advantage of using a modern desktop chip is that such chips already have documented support to allow software to control power states and monitor chip energy and temperature.

Testbed implementation. We engineer the thermal environment of our testbed such that it can sustain indefinitely (without overheating) a single-core running at the lowest configurable frequency, but all higher power operating points are not sustainable and hence represent various intensities of sprinting. We build the testbed from a system with an Intel Core i7 2600 quad-core "Sandy Bridge" chip. We remove the heat sink and place a small variable-speed fan several inches above the exposed package, as shown in


(a) Testbed setup

(b) Package internals

Figure 1. Testbed setup and package cut-away.

Figure 1(a). By tuning the fan speed, we can adjust the testbed's sustainable TDP. For this chip, single-core operation at 1.6 GHz (the lowest user-selectable frequency/voltage setting) draws 9.5 W. We tune the fan speed such that under sustained execution, die temperature saturates at 75° C (just under the rated junction temperature limit of 78° C). Although we had hoped to demonstrate sprinting using purely passive cooling, unfortunately, operating even a single core at minimum configurable frequency/voltage dissipates more power than can be sustained without a fan.

To control sprinting, we disable the chip's own frequency management and instead explicitly control the chip's operating point using the software ACPI interface to enable/disable cores and select frequency/voltage. With all cores executing at highest configurable frequency (3.2 GHz), this chip draws approximately 50 W. This setting represents our most intense sprinting mode (5× higher power than sustainable operation). In our experimental results, we use the chip's energy measurement facility to report package-level energy consumption and the on-die temperature sensors to report die temperature. We validated these energy readings by calibrating against measured wall-socket power. After each experiment, we wait until the temperature returns to the idle temperature before conducting the next experiment. Each experiment is repeated multiple times; we present average results over these runs.

Thermal capacitance from the integrated heat spreader. Sprinting leverages thermal capacitance to support brief operation at modes with unsustainable power dissipation. Like a mobile system, the testbed has no heatsink that would serve as a natural source of thermal capacitance for sprinting to exploit. It does, however, have an integrated heat spreader (IHS), which is a copper plate inside the package that is attached directly to the die via a thin thermal interface material, as illustrated in Figure 1(b). The traditional role of the IHS is to spread the heat: (*i*) on the die (to reduce hot spots) and (*ii*) from the die to the entire top of the package (to facilitate cooling), so the IHS is typically larger than the die.

The copper IHS provides sufficient thermal capacitance for sprinting. Given its dimensions (32 mm × 34 mm × 2 mm [4]) and the density (8.94 g/cc) and specific heat of copper (0.385 J/gK), we estimate its total heat capacity to be 7.5 J/K. When the system idles, its temperature settles at roughly 50° C. Thus, during sprinting, for a temperature swing of 25° C to a maximum temperature of 75° C, the IHS can store 188 J of heat. Therefore, in theory, the IHS present in this off-the-shelf package enables 4.7 s of sprinting at 40 W over sustainable TDP (50 W total). As we will show later, because lateral heat spreading is not instantaneous with the IHS, the actual achievable sprint duration at maximum intensity is somewhat less than this simple calculation suggests. We explore extending this sprint duration by augmenting the testbed with a phase change heat sink to increase thermal capacitance in Section 5.

The testbed relies on the thermal capacitance of the heat spreader for sprinting, and chips in mobile devices have typically not employed heat spreaders. However, the Apple A5X chip used in the third-generation iPad tablet does employ a heat spreader [13]. In addition, the dual-core A5X has a die size of 162 mm^2, which is nearly as large as the 216 mm^2 die of the quad-core Core i7 used in the testbed (albeit on 45 nm for the A5X vs. 32 nm for the Core i7). Perhaps the largest difference is that the peak power draw of the A5X chip is nowhere near the 50 W or more of the Core i7. Correspondingly, both the idle power and the power of using just one core of the Core i7 is substantially higher than the A5X. Based on this comparison and process scaling trends [11, 18, 20, 48, 52], a future mobile chip with more cores than can sustainably operate within the thermal constraints of a mobile device seems entirely plausible.

Effects not captured in the testbed. In addition to the thermal effects captured in the testbed (which are the focus of this paper), sprinting also places requirements on energy supply, including the voltage regulator, sufficient pins to carry the necessary peak electrical current, and power supply (which would be a battery in a mobile device). Prior work discusses these challenges and potential solutions [45], including ultra-capacitor/battery hybrids to meet peak current demands [39, 42, 44]. In our testbed, the electrical supply is already provisioned to sustain the chip's peak power, and hence is a non-issue. We do not address the electrical challenges or reliability/wearout implications of sprinting in this paper.

3. Unabridged Sprints

Computational sprinting is relatively easy to manage when sprints are *unabridged*, that is, when the parallel work can be completed entirely during the sprint without exhausting the system's thermal capacitance. This section explores the responsiveness and energy efficiency of unabridged sprints; we turn to the more complex case of truncated sprints in Section 4.

3.1 Responsiveness Benefits of Sprinting

Our testbed is able to sprint due to the thermal capacitance available in the copper mass of the integrated head spreader. Section 2 provides a back-of-the-envelope estimate for this capacitance as 188 J, which suggests a maximum sprint duration of 4.7 s at peak intensity. We compare this estimate against repeated trials of sprinting with all four cores at maximum configurable frequency (3.2 GHz) to establish the actual maximum sprint duration at peak. This operating mode dissipates approximately 5× the power of the sustainable single-core 1.6 GHz mode (50 W vs. 9.5 W) and can potentially improve responsiveness by 8× (2× frequency and 4× cores). For each trial, we allow the system to cool to a nominal of approximately 50° C and sprint until the system reaches our safe temperature threshold of 75° C as measured by internal on-die temperature sensors.

Kernel	Description
`sobel`	Edge detection filter; parallelized with OpenMP
`disparity`	Stereo image disparity detection; adapted from [51]
`segment`	Image feature classification; adapted from [51]
`kmeans`	Partition-based clustering; parallelized with OpenMP
`feature`	Feature extraction (SURF) from [14]
`texture`	Image composition; adapted from [51]

Table 1. Parallel workloads used to evaluate sprinting.

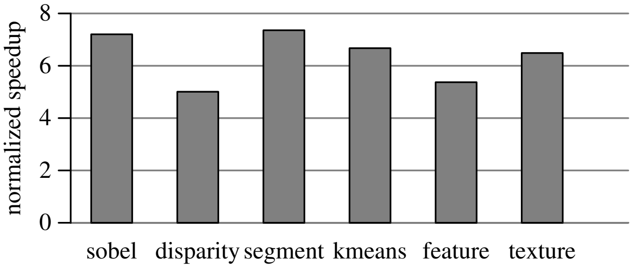
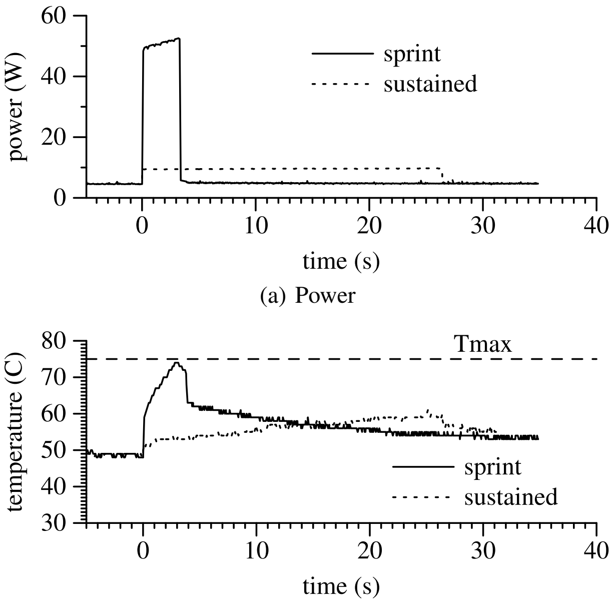


Figure 2. Speedup benefits of unabridged sprinting using four cores at 3.2 Ghz over a single-core 1.6 Ghz baseline.

We find that the testbed can sprint for at most three seconds at maximum intensity. The actual sprinting capability of the testbed is lower than the total thermal capacity of the bulk-copper heat spreader suggesting a delay in lateral heat spreading [34]—the temperature rise during sprints is so rapid that significant temperature gradients persist. Given the thermal conductivity of copper (between 300 and 400 W/mK), heat will spread only 16 mm to 19 mm during a 3 s sprint, which is insufficient to reach the corners of the heat spreader from the center of the die. As we show later in our study of truncated sprints, reducing sprint intensity allows more time for heat to spread, allowing more thermal capacitance to be utilized before the critical temperature is reached.

To explore the practical benefits of unabridged sprints in real workloads, we configure the work size for our parallel workloads (see Table 1) to complete before the thermal limit is reached. Figure 2 shows the responsiveness benefits of maximum-intensity sprints (four cores at 3.2 GHz) in terms of speedup over the sustainable single-core 1.6 GHz baseline. On average, sprinting provides a 6.3× benefit over the baseline. In essence, sprinting allows this system to complete in just a few seconds what would have taken fifteen seconds or more if constrained to operate only in sustainable (non-sprinting) mode.

To better understand the testbed's thermal behavior, Figure 3 shows its power and temperature response over time. Each graph has two lines, which correspond to the same computation being performed in sustainable (single-core) and maximum-intensity sprint mode for the `sobel` workload. At time zero (on the x-axis), the power in sustainable mode (dotted line) rises from the idle-draw of 4.5 W to 9.5 W (Figure 3(a)), causing a gradual increase in temperature (Figure 3(b)); the computation completes after 26 s, at which time the power level drops back to idle and the temperature begins to fall. Under sprinting (solid line), power jumps initially to 50 W, which causes the temperature to rise precipitously. In fact, the temperature rise is so large that the leakage power of the chip grows over the duration of the sprint, causing power to increase further to a peak of 55 W. After a few seconds, the computation completes, the chip returns to idle, and temperature falls quickly (although not as quickly as it rose).



(a) Power

(b) Temperature

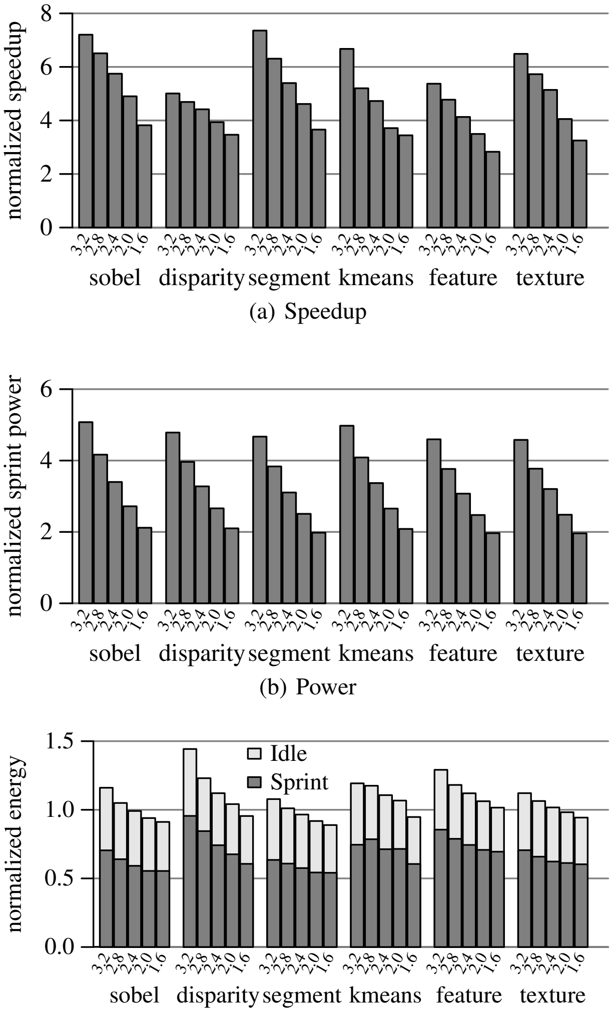
Figure 3. Chip power and thermal behavior for sustained and sprinting operation.

3.2 Energy Impacts of Sprinting

Energy consumption is a critical metric especially for battery-powered devices, so sprinting may not be an attractive option if it uses more energy than slower sustained execution. Perhaps counter-intuitively, however, we find sprinting can actually result in net gains in energy efficiency by (*i*) amortizing the fixed uncore power consumption over a larger number of active cores and (*ii*) capturing race-to-idle effects. Our conclusions stand in contrast to our previously published predictions from simulation-based analysis of computational sprinting [45], which neglected uncore power and hence suggested that sprinting might at best be energy neutral.

When considering energy implications, there are two distinct knobs available for sprinting: the number of cores and the clock frequency. Increasing the clock frequency requires increasing voltage as well, leading to a super-linear increase in energy consumption. In contrast, we find that increasing the number of cores results in a sub-linear energy increase, consistent with previously published estimates [6]. For the chip in our testbed, all but one core can be disabled (clock and power gated) but the last-level cache and ring interconnect are always powered and active unless the system has been suspended to DRAM. In this chip, this background power draw is substantial and largely independent of the number of active cores. Thus, activating all four cores requires only 2× more power than single-core operation. For any platform in which background/static power dissipation is substantial, using more cores will improve energy efficiency and responsiveness for unabridged sprints by amortizing these fixed overheads (provided the workload performance scales with the number of cores). On our testbed, four-core sprints dominate configurations that use fewer cores, hence, we omit such configurations from our results.

Whereas our responsiveness analysis in the preceding section considered only the most intense possible sprint, incorporating all cores at maximum frequency/voltage, because of the super-linear power increase of voltage/frequency scaling, we also consider sprinting with lower frequencies to improve energy efficiency. Figure 4 shows the responsiveness, power, and energy implications of sprinting with all four cores at various frequency/voltage lev-



(a) Speedup

(b) Power

(c) Energy

Figure 4. Speedup, power, and energy (normalized to the one-core 1.6 GHz sustainable baseline) for four cores across frequencies.

els. The left-most bar in each group of Figure 4(a) shows the same speedup (6.3× on average) with a maximum-intensity four-core 3.2 GHz sprint that was presented earlier in Figure 2. The corresponding bars in Figure 4(b) show the 5× power increase for such a sprint. As the frequency is lowered, the responsiveness improvements decrease, but power decreases relatively more.

Figure 4(c) shows the energy implications of each sprint. We apportion energy into two components. The lower portion of the bar depicts the energy consumed during the sprint itself; because of the non-linear relationship between performance and power under frequency/voltage scaling, higher frequency sprints consume more energy. The upper portion of each bar depicts the energy consumed as a result of idle power dissipated after the end of the sprint until the work would have completed under sustainable single-core execution. By measuring energy consumption over the period of the slower, sustainable execution, we facilitate a fair efficiency comparison.

Ignoring idle energy for a moment, we see that even sprinting at maximum intensity reduces the energy consumed during the computation. The background power conserved by finishing sooner more than compensates for the super-linear cost of voltage scaling, reducing energy up to 30%. At lower frequencies, this potential energy savings grows to nearly 40%. Similar race-to-idle effects have been observed previously [5, 7, 21, 22, 33, 36, 40].

Unfortunately, chips continue to dissipate power even when idle. By finishing earlier, the chip must idle for longer [40]; thus we must also consider the top segment of each bar, which adds the energy consumed during the additional idle time. For the 3.2 GHz four-core sprint, once the idle energy is included, the total energy is 21% higher than the sustainable baseline. However, when we consider the lowest-frequency four-core sprint, we find an energy efficiency gain of 6% even when including idle energy. Hence, when sprint intensity is selected appropriately, sprinting can improve energy efficiency as well as responsiveness even on today's chips. Finally, we note that, even though the chip idle power is already less than one tenth of its peak, there is still ample motivation to optimize idle power further: the energy efficiency advantages of sprinting grow rapidly as idle power vanishes. With several emerging mobile architectures seeking to aggressively reduce idle power (*e.g.*, NVIDIA Tegra 3's vSMP/4-plus-1 [41] and ARM's big.LITTLE multicores [24]), we see substantial potential for sprinting as an energy saver as well as a responsiveness enabler.

4. Truncated Sprints

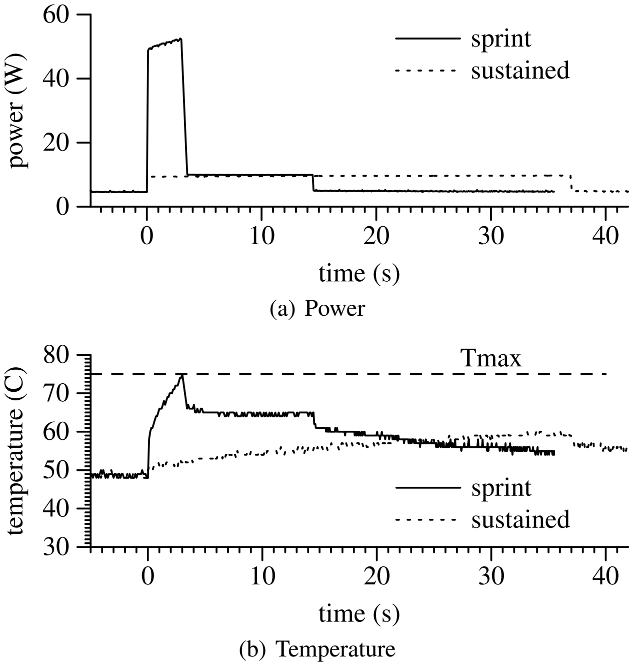
Ideally, all sprints would be unabridged, completing before available thermal capacitance is exhausted. However the system must avoid overheating for computations that cannot be completed entirely within a sprint while aiming to preserve some of the responsiveness benefits of sprinting.

This section first describes the software for truncating sprints to avoid overheating by throttling frequency and disabling all but one core. Next, we observe that for some workloads, the naive approach to completing work after truncation, *i.e.*, migrating threads to be multiplexed on a single core, can result in significant degradation in performance and energy efficiency. We explore mitigating these effects using a sprinting-aware task-based parallel runtime. Finally, we motivate *sprint pacing* by showing that the most responsive mode of sprinting is not always maximum-intensity sprints, but instead is sprinting at the most intense rate that still allows the sprint to remain unabridged.

4.1 Implementing and Evaluating Sprint Truncation

The testbed software monitors die temperature using a thread to query the on-die temperature sensor every 100 ms. Although implemented as a user-level thread in our testbed prototype, ultimately this functionality would likely be integrated into the operating system's dynamic thermal management facility. When the die temperature reaches T_{max}, the software truncates the sprint by: (*i*) pinning all threads to a single core to force the operating system to migrate the threads, (*ii*) disabling the now-idle cores, and (*iii*) changing the frequency of the single core to the lowest configurable frequency. The testbed software implements these steps using system calls and the standard Linux ACPI interface.

To explore the behavior of sprint truncation, we reconfigure the workloads to run longer so that the computation exceeds the available thermal capacitance for sprinting. Figure 5 shows the impact of sprint truncation on the power and temperature of the chip over time for both sustained execution and a truncated sprint. As before, temperature rises sharply when sprinting begins from the idle state at time zero. Once the temperature reaches the T_{max} value of 75° C, the runtime system invokes sprint truncation, which results in the abrupt drop in power from 55 W to 9.5 W. In response, the temperature stops rising, as the system's power consumption now matches the rate of cooling dissipation. In fact, the temperature drops initially as the heat spreads throughout the die, package, and surrounding components. After truncation, the computation continues in sustainable mode with all threads multiplexed on a



(a) Power

(b) Temperature

Figure 5. Power and thermal response for truncated sprints.

single active core at minimum frequency. The system's thermal response matches that of the sustained computation during this interval. When the computation completes, the remaining core becomes idle, and the chip begins to cool.

Performance and energy penalties of sprint truncation. To evaluate the impact of sprint truncation, we vary the length of each computation and measure responsiveness across sprint intensities. In Figure 6, each group of bars shows the execution time (Figure 6(a)) or energy (Figure 6(b)) for maximum-intensity sprinting for varying computation lengths normalized to a non-sprinting system. The segments of each bar indicate the fraction of time spent in sprint (bottom segment), sustained (middle segment), and idle modes (top segment). Unsurprisingly, as the computation length increases beyond the sprint capacity, larger and larger fractions of time are spent computing in sustained mode. Correspondingly, the impact of sprinting on execution time (and thus responsiveness) and energy is smaller for longer computations as less time is spent sprinting. One seeming anomaly is that truncated sprinting is actually slower than the sustained baseline for two workloads (`feature` and `texture`), which we address next.

4.2 Mitigating Overheads of Truncated Sprints

Sprint truncation results in all active threads being multiplexed on the single remaining core, which leads to a net slowdown in some workloads relative to the sustainable baseline (Figure 6(a)). Although it is expected that long-running computations would receive little benefit from an initial sprint, the observed degradation is highly undesirable as ideally sprinting would "do no harm" to long-running computations. The observed degradation in these workloads is a result of multiplexing all threads on a single core. The resulting oversubscribed system is prone to known pathologies from contention on synchronization, load imbalance, convoying, and frequent context switches [9, 25, 28, 29, 50].

Demonstrating the penalty of oversubscription. To demonstrate the performance penalty of oversubscription, Figure 7 shows the performance impact of spawning N threads but pinning them to a single core. As the amount of oversubscription increases, so does the penalty. Although most of the workloads are not sensitive to oversubscription (omitted for brevity), the effect is particularly pronounced in `texture`, where we see a penalty of over 2.5× for a four-to-one oversubscription ratio. The penalty is as high as 5.8× for 16 threads.

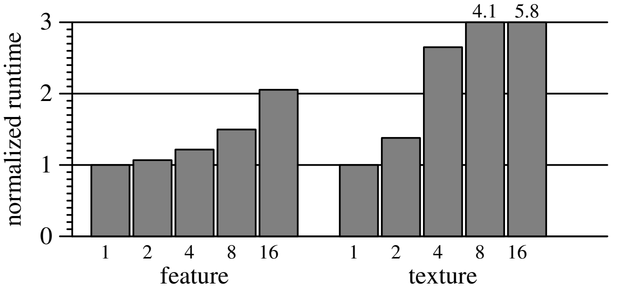
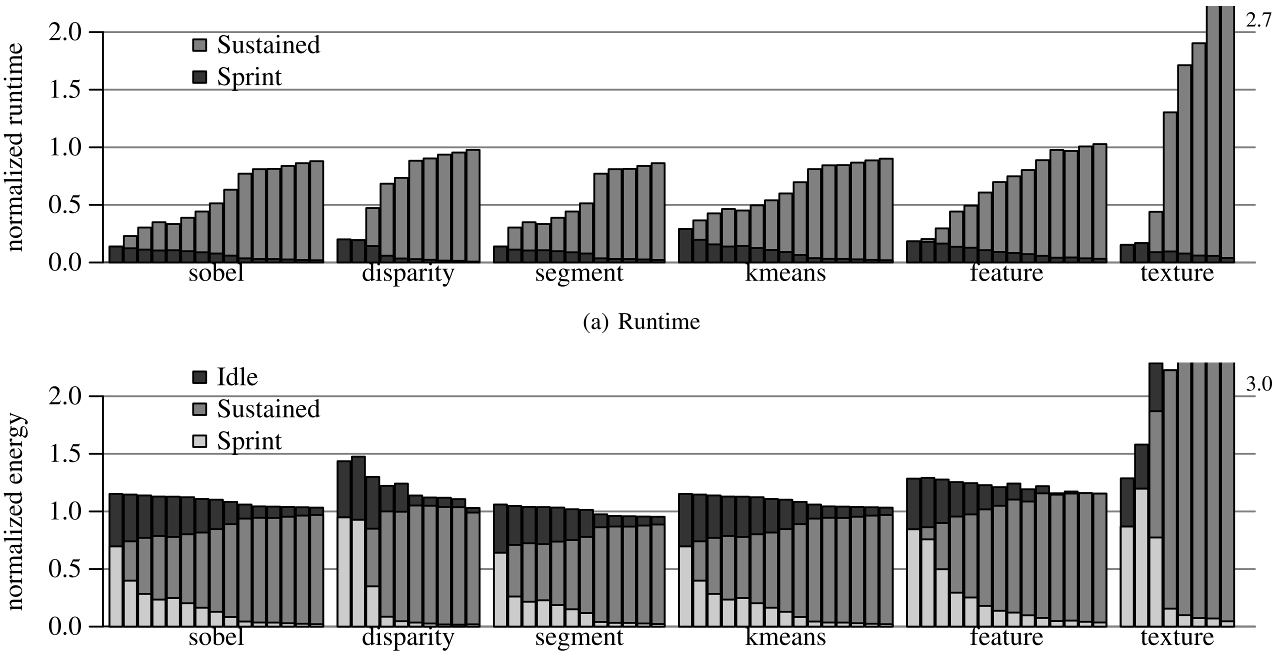


Figure 7. Runtime penalty from oversubscription with increasing numbers of threads on a single core.

Conventional approaches to mitigating the penalty of oversubscription. This well-known phenomenon has several typically prescribed mitigation approaches. One approach—avoiding the problem of oversubscription by spawning only as many threads as cores—is not applicable because sprint truncation changes the number of available cores while the computation is executing. Another mitigating approach is to tailor shared-memory synchronization primitives (locks and barriers) to yield the processor instead of busy waiting. We explored various implementations and our reported results already include user-level synchronization primitives that `yield()` after spinning for 1000 cycles. These primitives reduced the prevalence of the oversubscription penalty, but extra context switches are still required and the penalty remains for two workloads due to their frequent use of barrier synchronization. Another approach is to have programs dynamically adjust the number of active threads [50], which is the approach we build upon here.

Sprint-aware task-based parallel runtime to mitigate oversubscription penalties. Efficient sprint truncation requires an efficient mechanism to dynamically change the number of active software threads. To provide such a mechanism, we look toward task-queue based worker thread execution frameworks [1, 10, 16, 19, 30, 50], in which applications are decomposed into tasks and the program is oblivious to the actual number of worker threads. In such frameworks, the tasks created by the application are then assigned to thread queues. Worker threads first look for tasks to execute in their local task queue. Upon the absence of a local task, workers "steal" tasks from other threads' task queues. The core-oblivious nature of the task-based model, coupled with its automatic load balancing via task stealing facilitates dynamically changing the number of worker threads. Reducing the number of threads is as simple as having a worker thread go to sleep once it has completed its current task.

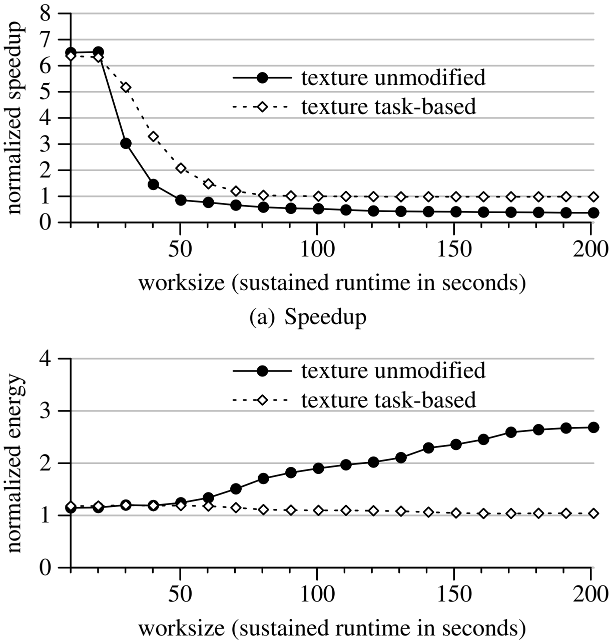
We mitigate oversubscription penalties with a sprint-aware task-stealing runtime framework (similar to Intel's Threading Building Blocks [1]). Before dequeuing another task to execute, a thread first queries a shared variable that indicates the number of worker threads that should be active. If the desired number of active threads is lower than the worker thread's identifier, the worker thread exits. Any pending tasks in that thread's queues will eventually be stolen and executed by the worker thread running on the single remaining core after a sprint is truncated. The same monitor thread that otherwise handles sprint termination is responsible for setting the desired number of active threads by writing to the shared variable. This task-based mechanism does not replace the existing thread migration and core disabling mechanism; that mechanism is still needed



(a) Runtime

(b) Energy

Figure 6. Runtime and energy spent during sprinting, sustained, and idle modes for 4-core sprints at 3.2 Ghz (normalized to the one-core 1.6 Ghz baseline.) Bars represent increasing computation lengths from left to right.



(a) Speedup

(b) Energy

Figure 8. Speedup and energy comparison of the unmodified threaded and task-based implementations of `texture`.

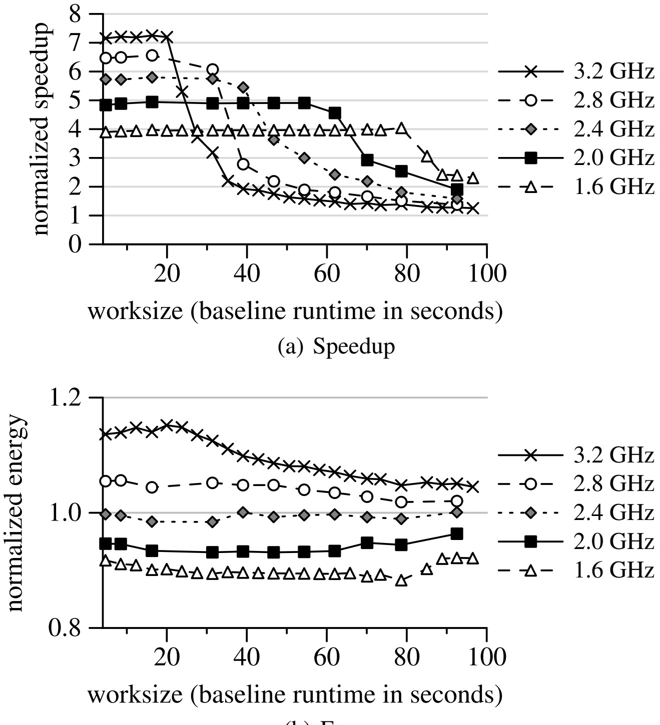
in case a thread is executing a long task, which must be suspended and migrated to avoid overheating. However, the task-based policy ensures that eventually all but one worker thread will be put to sleep, thus avoiding the oversubscription penalty for the remainder of the computation.

To evaluate the effectiveness of this approach, we create a variant of the `texture` workload rewritten for this task-based parallelism model. Figure 8 shows the sprint truncation behavior of the original multi-threaded version of `texture` and the task-based variant for various computation lengths. Both versions have similar responsiveness when the computation is short and the sprint is unabridged. However, whereas the performance of the original multi-threaded version of `texture` falls well below that of the sustainable baseline, the task-based `texture` variant converges to it. This experiment indicates that a sprint-aware task-based runtime can eliminate the inefficiencies of sprint truncation, allowing for robust "do no harm" sprinting.

4.3 Sprint Pacing

Section 3 concluded that for unabridged sprints, sprinting at maximum intensity is best to improve responsiveness. However, when maximum-intensity sprinting results in sprint truncation, the choice of sprint intensity is not so simple. Figure 9 shows responsiveness benefits over a sustainable baseline for four-core sprints across frequencies ranging from 3.2 GHz to 1.6 GHz. For short computations (the far left of the graph), maximum-intensity sprinting maximizes responsiveness; for large computations, the responsiveness is no better than sustainable execution. However, for intermediate computation lengths, the optimal sprinting mode is not always maximum sprint intensity. In such cases—just as in human runners—it is better to sprint at a slower pace for longer than to sprint at maximum pace for an extremely short duration. This observation motivates the need for a *sprint pacing policy*.

Benefits of paced sprinting. To better understand the opportunity for sprint pacing, consider the difference in maximum sprint duration for four cores at 3.2 GHz vs. 1.6 GHz, illustrated with power and temperature curves in Figure 10. The responsiveness advantage due to doubling frequency is 2× at best. However, the maximum sprint duration at 1.6 GHz is 6.3× longer than the 3.2 GHz



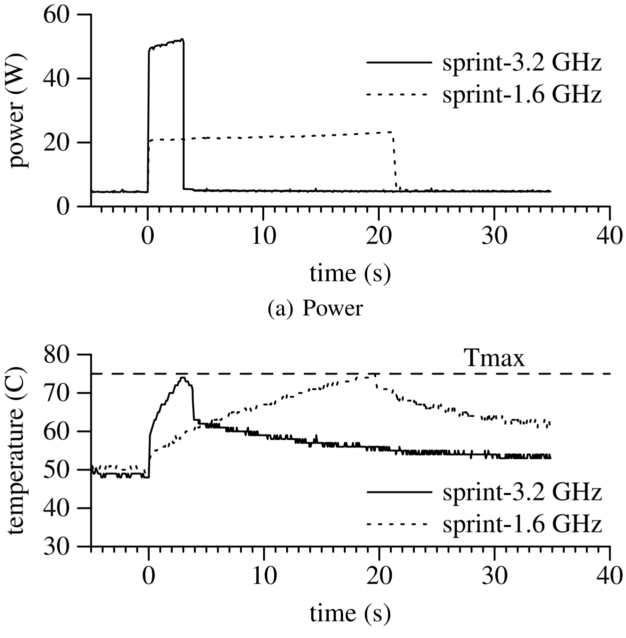
(a) Speedup

(b) Energy

Figure 9. Speedup and energy versus size of computation for sprinting with four cores at different differences.

sprint, implying that a 1.6 GHz sprint can complete over 3× more work. The less intense sprint completes more work for three reasons. First, lowering frequency and voltage results in a more energy efficient operating point, so the thermal capacitance consumed per unit of work is lower. Second, the longer sprint duration allows more heat to be dissipated to ambient during the sprint. Third, as we discussed previously, maximum intensity sprints are unable to fully exploit all thermal capacitance in the heat spreader because the lateral heat conduction delay to the extents of the copper plate is larger than the time for the die temperature to become critical. By sprinting less intensely, more time is available for heat to spread and more of the heat spreader's thermal capacitance can be exploited.

Toward a sprint pacing policy. The most critical impact on sprint pacing policy is the length of the computation, and we envision two general approaches to sprint pacing. The first approach is *predictive sprint pacing* in which the length of the computation is predicted to select a near-optimal sprint pace. Such a prediction (*e.g.*, [27]) could be performed by the hardware, operating system, or with hints from the application program directly. In the absence of such a prediction, an alternative approach is *adaptive sprint pacing* in which the pacing policy dynamically adapts the sprint pace to capture the best-case benefit for short computations, but moves to a less intense sprint mode to extend the length of computations for which sprinting improves responsiveness. Figure 11 shows the behavior of a simple adaptive sprint policy that sprints at full intensity until half of the thermal capacity is consumed, after which it switches to less intense and more power-efficient sprints by keeping all four cores active but throttling their frequency to 1.6 GHz. As shown in the graph, this policy captures the benefits of sprinting for short computations but maintains some responsiveness gains for longer computations. Although the dynamic policy falls short of an *a priori* selection of the best sprint intensity for some computation lengths, it is robust in that it provides benefits over a larger range of computation lengths.



(a) Power

(b) Temperature

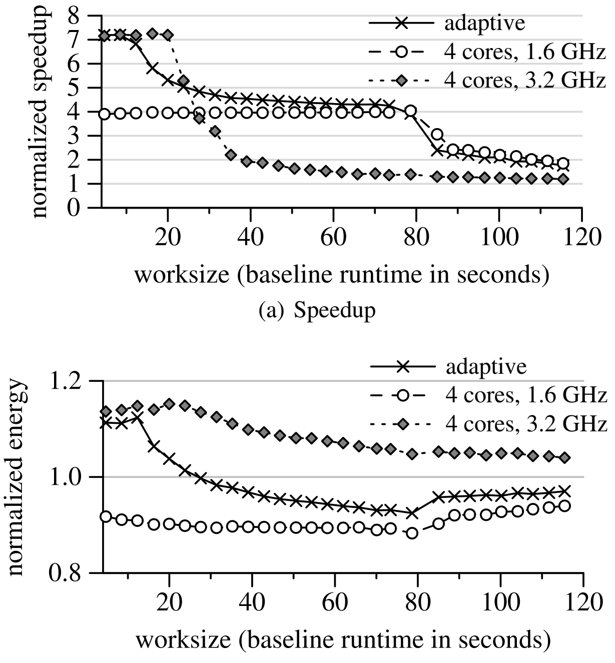
Figure 10. Power and thermal response for sprinting with four cores at 1.6 GHz.

Other parameters that may impact sprint pacing policies. The optimal sprint pace is potentially impacted by other factors. Although the most basic factor is the length of the computation, other factors include the performance and power impact of both the clock frequency and the number of cores [31, 32, 43]. For example, a workload that has poor parallel scaling may benefit more from higher frequency than additional cores. In our four-core testbed with workloads that scale well, we found such effects were not significant, but they will likely become more critical in the future as the number of cores on a chip increases.

5. Extending Sprints using Latent Heat

Thus far, this paper has examined policy approaches to mitigating the impacts of limited sprint duration. A complementary approach to increase sprinting effectiveness is to engineer a system that supports longer sprints by including more thermal capacitance. Instead of leveraging only the specific heat of conventional materials, prior work proposed using the latent heat of a phase change material (PCM) to add thermal capacitance to a sprinting system [45]. Latent heat has the advantage that it can absorb substantial heat without a change in temperature. Whereas each gram of copper in the heat spreader can absorb 11.5 J over a 30° C rise, many phase change materials used for heat storage can absorb 200 J per gram or more [55].

To test the potential of phase change materials to extend sprint duration, we perform proof-of-concept experiments using our testbed. We are unaware of a readily available PCM designed to meet the needs of computational sprinting (e.g., stability over numerous rapid thermal cycles with a melting point within the relevant temperature range), so we test paraffin (BW-40701 from BlendedWaxes, Inc.) which has a melting point of 54° C. Because paraffin has poor thermal conductivity (0.2 W/mK), we infuse it in 0.9 g of Doucel aluminum foam (bulk thermal conductivity of 5 W/mK). To prevent leakage, we enclose the paraffin/foam structure in a 4.2 cm × 4.2 cm × 0.3 cm box of 0.013 cm thick copper weighing 4 g. We then mount this enclosure on the processor socket



(a) Speedup

(b) Energy

Figure 11. Speedup and energy for sprinting based on an adaptive allocation of sprint budget between the most responsive and most energy efficient schemes.

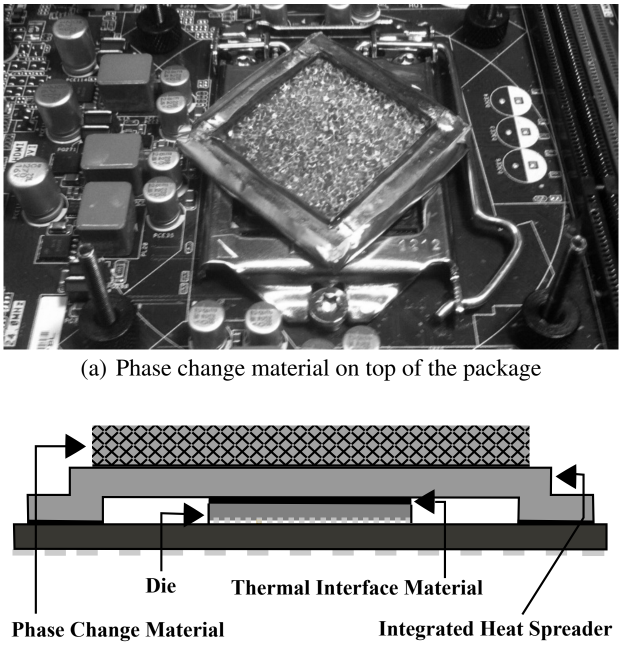
using screws to provide firm attachment and improve interfacial heat transfer. Figure 12 illustrates our setup.

Figure 13 shows the thermal response of sprinting with four cores at 1.6 GHz on this modified testbed, isolating the effects of each of the labeled components. Adding the copper container and aluminum foam alone (labeled empty foam) increases thermal capacitance due to the additional specific heat and nearly doubles the baseline (air) sprint duration (37 s vs. 20 s). With the addition of 4 g of PCM (wax), the testbed can sprint for 120 s—6× over the baseline. The flattening in the PCM temperature curve is a consequence of the PCM melting. Given the latent heat of paraffin wax (200 J/g), 4 g of such material can absorb about 800 J of heat when melting, corresponding to an additional 40 s of sprint duration at 20 W. However, the observed sprint extension exceeds this estimate largely due to the additional heat dissipated to the ambient over this duration, and to a smaller extent, due to the specific heat of the wax.

To further distinguish the contribution of latent heat from specific heat, we replace the PCM with an equal weight of water (and a plastic cap to prevent the water from evaporating), and observe a sprint duration of 50 s. As the specific heat of water (4.2 J/gK) is higher than that of paraffin (2 J/gK), we conclude that the latent heat of the PCM indeed accounts for the substantial sprint extension. Although our experiments confirm that using PCM can be an effective approach for extending sprint duration, significant opportunity remains for engineering more effective PCM materials and composites, especially if incorporated directly within the package.

6. Sprint-and-Rest for Extended Computations

Whereas introducing additional thermal capacitance facilitates longer sprints, maximum sprint duration remains finite. In the previous sections, we have analyzed sprint policy as a function of workload size while considering only a single sprint. In this section, we consider whether sprinting can also be beneficial for extended (even indefinite) computations.



(a) Phase change material on top of the package

(b) Cross-section of phase change material on the package

Figure 12. Testbed augmented with phase-change material.

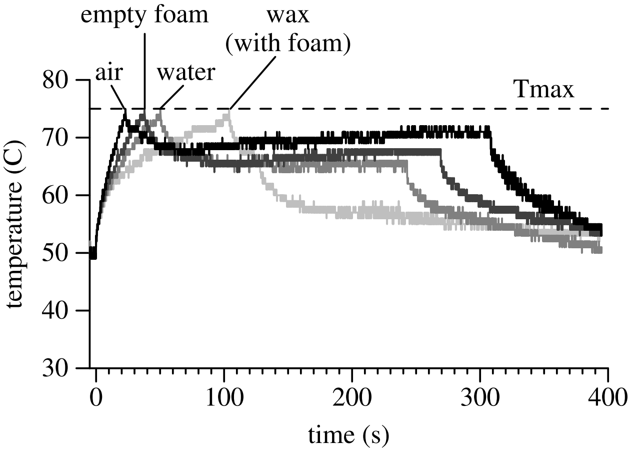
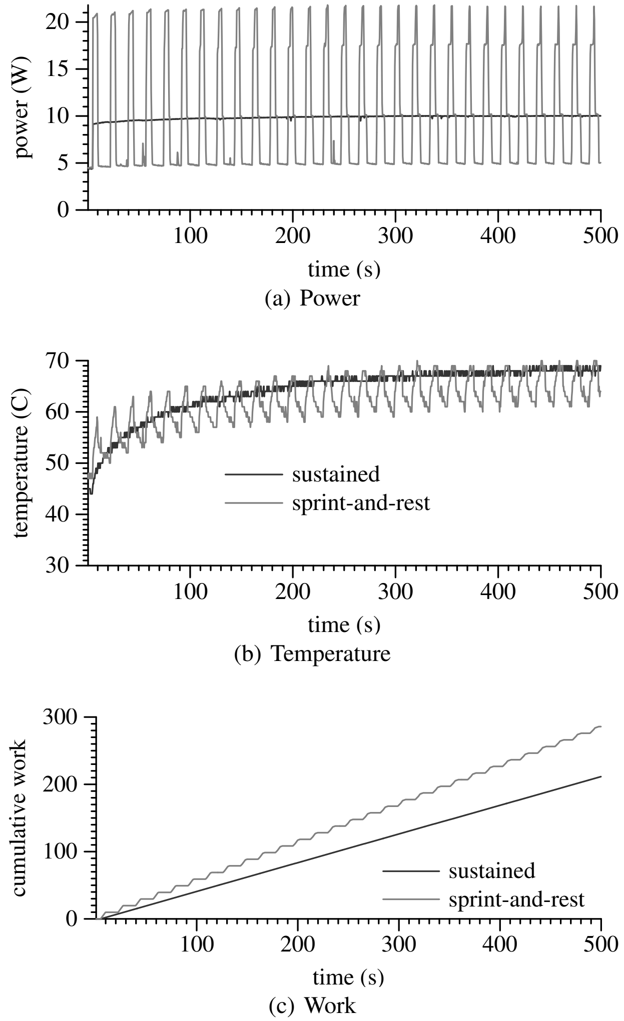


Figure 13. Comparison of sprinting thermal response with and without PCM.

Over the long run, the average power consumption of a platform is constrained by the heat dissipation of the cooling solution (*i.e.*, the platform's TDP). The obvious way to execute a long-running computation is to select a sustainable operating mode that consumes less power than the TDP (in which case the chip can operate indefinitely). However, in a sprint-enabled system, one can also consider an operating regime that alternates between sprint and rest periods. Provided (*i*) the sprint periods are short enough to remain within temperature bounds and (*ii*) the rest periods are long enough to dissipate the accumulated heat, such a sprint-and-rest mode of operation is also sustainable indefinitely. That is, sprint-and-rest is sustainable as long as the average (but not necessarily instantaneous) power dissipation over a sprint-and-rest cycle is at or below the platform's sustainable power dissipation.



(a) Power

(b) Temperature

(c) Work

Figure 14. Comparison of power, temperarature, and cumulative work done with sprint-and-rest and sustained computation.

We contrast conventional sustainable operation using a single-core at 1.6 GHz and sprint-and-rest operation with four cores at 1.6 GHz. For the sprint-and-rest mode to be sustainable, its average power draw must be at or below the 9.5 W dissipation of the single-core mode. On our testbed, a four-core 1.6 GHz sprint consumes 20 W, while idling consumes 4.5 W. Hence, the maximum sustainable sprint-to-rest ratio is 1:2.1. We conservatively choose a ratio of 1:2.5, with a sprint duration of 5 s followed by a rest period of 12.5 s. We repeat this cycle over several minutes and measure temperature fluctuations, power, and the total work completed.

Counterintuitively, we found that this sprint-and-rest mode provides *higher* performance despite marginally lower average power consumption than the sustainable baseline. Figure 14 shows power, temperature, and cumulative completed work over several minutes. The power plot (Figure 14(a)) shows the staccato instantaneous power pattern of sprint-and-rest operation. The temperature plot (Figure 14(b)) reveals temperature increases for both modes—with sprint-and-rest following a sawtooth pattern—but both modes converge to a temperature below the 75° C threshold. Careful examination of the power plot reveals a slight upward trend for the power consumption of both sustained and sprint-and-rest execution, due to temperature-dependent leakage. The cumulative work plot (Figure 14(c)) demonstrates the performance advantage of sprint-and-rest; despite the stair-step pattern of forward progress, sprint-and-rest completes work at an average rate 35% higher than single-core sustained. Hence, sprint-and-rest provides both greater performance and better energy-efficiency than steady sustained operation.

Sprint-and-rest outperforms TDP-constrained sustained operation because the instantaneous energy efficiency of quad-core operation is better than single-core operation; operating all four cores provides quadruple the performance at double the power. This benefit arises because quad-core operation amortizes the fixed power costs of operating the chip over more useful work. Sprint-and-rest will provide a net efficiency gain whenever the instaneous energy-efficiency ratio of sprint vs. sustainable operation exceeds the sprint-to-rest time ratio required to cool. The advantages of sprint-and-rest grow if the idle power of the chip is reduced. We expect that similar observations may hold for chips that provide other kinds of performance-power assymetry, for example, due to heterogeneous cores. On the other hand, repeated thermal cycling introduced by sprint-and-rest can affect the reliability of the chip and potentially cause thermal stress in packaging components like solder bumps. We leave an analysis of reliability to future work.

7. Conclusions

Finding ways to deliver value to customers from dark silicon may be one of the defining challenges of the next decade of computer architecture research. Computational sprinting offers one such value proposition for bursty interactive applications by transiently activating dark silicon to deliver performance when end users need it most—when they are waiting for their device to respond.

Whereas prior work argued for the feasibility and potential of computational sprinting in theory, in this paper we reduce it to practice through experimentation on a concrete hardware/software sprinting testbed. We have shown that even though our testbed is constructed with an existing chip that was not designed with sprinting in mind, sprinting can provide not only substantial gains in responsiveness, but in fact also provides net energy efficiency gains by racing to idle. Even for extended computations, we find that a thermally constrained sprint-enabled chip achieves better performance through sprint-and-rest operation rather than sustained execution. The central insight underlying these counterintuitive results is that chip energy efficiency is maximized by activating all useful cores—disregarding thermal limits—to best amortize the fixed costs of operating at all. Moreover, our results provide ample motivation for chip designers to further optimize idle power; although many chips already achieve 10-to-1 ratios between peak and idle power, our results indicate that further overall energy efficiency gains of nearly 40% could be achieved by driving down idle power.

Our study demonstrates the synergy between task-based work-stealing parallelism and sprinting; by dissociating parallel work from specific threads, we give the runtime the freedom it needs to manage sprint pacing and avoid oversubscription penalties for truncated sprints. We have performed an investigation of sprint pacing, demonstrating the benefits of adaptive pacing, but anticipate rich opportunities for further innovation as chips scale in heterogeneity and number of cores. Finally, we have shown the first experimental results that phase change materials can indeed extend sprint durations, which may open a myriad of avenues for innovation in packaging and thermal management.

Acknowledgments

This work was supported in part by NSF grants CCF-0644197, CCF-0815457, CCF-1161505 and CCF-1161681. The authors would like to thank Yatin Manerkar and the anonymous reviewers for their feedback.

References

[1] Threading Building Blocks.
URL `http://threadingbuildingblocks.org`.

[2] Nokia Point and Find, 2006. URL `http://www.pointandfind.nokia.com`.

[3] Google Goggles, 2009. URL `http://www.google.com/mobile/goggles`.

[4] 2nd Generation Intel Core Processor Family Desktop and Intel Pentium Processor Family Deskop, and LGA1155 Socket, 2011. URL `http://www.intel.com/content/dam/doc/guide/2nd-gen-core-lga1155-socket-guide.pdf`.

[5] S. Albers and A. Antoniadis. Race to Idle: New Algorithms for Speed Scaling with a Sleep State. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, 2012.

[6] O. Azizi, A. Mahesri, B. C. Lee, S. J. Patel, and M. Horowitz. Energy Performance Tradeoffs in Processor Architecture and Circuit Design: A Marginal Cost Analysis. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, June 2010.

[7] P. Bailis, V. J. Reddi, S. Gandhi, D. Brooks, and M. I. Seltzer. Dimetrodon: Processor-level Preventive Thermal Management via Idle Cycle Injection. In *Proceedings of the 48th Design Automation Conference*, June 2011.

[8] G. Blake, R. G. Dreslinski, T. Mudge, and K. Flautner. Evolution of Thread-Level Parallelism in Desktop Applications. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, June 2010.

[9] M. Blasgen, J. Gray, M. Mitoma, and T. Price. The Convoy Phenomenon. *ACM SIGOPS Operating Systems Review*, 13, April 1979.

[10] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou. Cilk: An Efficient Multithreaded Runtime System. In *Proceedings of the 5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP)*, July 1995.

[11] S. Borkar and A. A. Chien. The Future of Microprocessors. *Communications of the ACM*, 54(5):67–77, 2011.

[12] K. Chakraborty. *Over-provisioned Multicore Systems*. PhD thesis, University of Wisconsin, 2008.

[13] Chipworks. The New iPad: A Closer Look Inside, Mar. 2012.
URL `http://www.chipworks.com/en/technical-competitive-analysis/resources/recent-teardowns/2012/03/the-new-ipad-a-closer-look-inside/`.

[14] J. Clemons, H. Zhu, S. Savarese, and T. Austin. MEVBench: A Mobile Computer Vision Benchmarking Suite. In *Proceedings of the IEEE International Symposium on Workload Characterization*, Sept. 2011.

[15] Computing Community Consortium. 21st Century Computer Architecture: A Community Whitepaper, Mar. 2012. URL `http://cra.org/ccc/docs/init/21stcenturyarchitecturewhitepaper.pdf`.

[16] G. Contreras and M. Martonosi. Characterizing and Improving the Performance of Intel Threading Building Blocks. In *Proceedings of the IEEE International Symposium on Workload Characterization*, Sept. 2008.

[17] B. Erol, E. Antunez, and J. J. Hull. PACER: Toward a Cameraphone-based Paper Interface for Fine-grained and Flexible Interaction with Documents. In *Proceedings of the International Symposium on Multimedia*, 2009.

[18] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger. Dark Silicon and the End of Multicore Scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, June 2011.

[19] M. Frigo, C. E. Leiserson, and K. H. Randall. The Implementation of the Cilk-5 Multithreaded Language. In *Proceedings of the SIGPLAN 1998 Conference on Programming Language Design and Implementation*, June 1998.

[20] S. H. Fuller and L. I. Millett. Computing Performance: Game Over or Next Level? *IEEE Computer*, 44(1):31–38, Jan. 2011.

[21] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy. Optimal Power Allocation in Server Farms. In *Proceedings of the 2009 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, June 2009.

[22] M. Garrett. Powering Down. *Queue*, 5(7):16–21, 2007.

[23] B. Girod, V. Chandrasekhar, D. M. Chen, N.-M. Cheung, R. Grzeszczuk, Y. Reznik, G. Takacs, S. S. Tsai, and R. Vedantham. Mobile Visual Search. *IEEE Signal Processing Magazine*, July 2011.

[24] P. Greenhalgh. Big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7: Improving Energy Efficiency in High-Performance Mobile Platforms, Sept. 2011.

[25] A. Gupta, A. Tucker, and S. Urushibara. The Impact of Operating System Scheduling Policies and Synchronization Methods of Performance of Parallel Applications. In *Proceedings of the 1991 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, May 1991.

[26] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki. Toward Dark Silicon in Servers. *IEEEMICRO*, 31(4):6–15, July 2011.

[27] C. J. Hughes, P. Kaul, S. V. Adve, R. Jain, C. Park, and J. Srinivasan. Variability in the Execution of Multimedia Applications and Implications for Architecture. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, July 2001.

[28] F. R. Johnson, R. Stoica, A. Ailamaki, and T. C. Mowry. Decoupling Contention Management from Scheduling. In *Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2010.

[29] L. I. Kontothanassis, R. W. Wisniewski, and M. L. Scott. Scheduler-Conscious Synchronization. *ACM Transactions on Computer Systems*, 15(1):3–40, Feb. 1997.

[30] D. Lea. A Java Fork/Join Framework. In *Proceedings of the ACM Java Grande 2000 Conference*, 2000.

[31] J. Li and J. F. Martínez. Power-Performance Considerations of Parallel Computing on Chip Multiprocessors. *ACM Transactions on Architecture and Code Optimization*, 2(4):397–422, Dec. 2005.

[32] J. Li and J. F. Martinez. Dynamic Power-Performance Adaptation of Parallel Computation on Chip Multiprocessors. In *Proceedings of the 12th Symposium on High-Performance Computer Architecture*, Feb. 2006.

[33] X. Li, Z. Li, F. David, P. Zhou, Y. Zhou, S. Adve, and S. Kumar. Performance Directed Energy Management for Main Memory and Disks. In *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 2004.

[34] Y. Li, B. C. Lee, D. Brooks, Z. Hu, and K. Skadron. CMP Design Space Exploration Subject to Physical Constraints. In *Proceedings of the 12th Symposium on High-Performance Computer Architecture*, Feb. 2006.

[35] G. Loudon, O. Pellijeff, and L. Zhong-Wei. A Method for Handwriting Input and Correction on Smartphones. In *Proceedings of the 7th International Workshop on Frontiers in Handwriting Recognition*, 2000.

[36] D. Meisner, B. T. Gold, and T. F. Wenisch. PowerNap: Eliminating Server Idle Power. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2009.

[37] R. Merritt. ARM CTO: Power Surge Could Create 'Dark Silicon'. *EE Times*, Oct. 2009.
URL `http://www.eetimes.com/electronics-news/4085396/ARM-CTO-power-surge-could-create-dark-silicon`.

[38] F. J. Mesa-Martinez, E. K. Ardestani, and J. Renau. Characterizing Processor Thermal Behavior. In *Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2010.

[39] A. Mirhoseini and F. Koushanfar. HypoEnergy: Hybrid Supercapacitor-Battery Power-Supply Optimization for Energy

Efficiency. In *Proceedings of the Conference on Design, Automation and Test in Europe*, Mar. 2011.

[40] A. Miyoshi, C. Lefurgy, E. Van Hensbergen, R. Rajamony, and R. Rajkumar. Critical Power Slope: Understanding the Runtime Effects of Frequency Scaling. In *Proceedings of the 2002 International Conference on Supercomputing*, June 2002.

[41] *Variable SMP (4-PLUS-1TM) A Multi-Core CPU Architecture for Low Power and High Performance*. NVIDIA, 2011.

[42] L. Palma, P. Enjeti, and J. Howze. An Approach to Improve Battery Run-time in Mobile Applications with Supercapacitors. In *34th Annual IEEE Power Electronics Specialist Conference*, volume 2, June 2003.

[43] S. Park, W. Jiang, Y. Zhou, and S. Adve. Managing Energy-Performance Tradeoffs for Multithreaded Applications on Multiprocessor Architectures. In *Proceedings of the 2007 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, June 2007.

[44] M. Pedram, N. Chang, Y. Kim, and Y. Wang. Hybrid Electrical Energy Storage Systems. In *Proceedings of the 2010 International Symposium on Low Power Electronics and Design*, 2010.

[45] A. Raghavan, Y. Luo, A. Chandawalla, M. C. Papaefthymiou, K. P. Pipe, T. F. Wenisch, and M. M. K. Martin. Computational Sprinting. In *Proceedings of the 17th Symposium on High-Performance Computer Architecture*, Feb. 2012.

[46] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissmann. Power Management Architecture of the 2nd Generation Intel Core Microarchitecture, Formerly Codenamed Sandy Bridge. In *Hot Chips 23 Symposium*, Aug. 2011.

[47] A. Shye, B. Scholbrock, and G. Memik. Into the Wild: Studying Real User Activity Patterns to Guide Power Optimizations for Mobile Architectures. In *Proceedings of the 42nd International Symposium on Microarchitecture*, Nov. 2009.

[48] M. B. Taylor. Is Dark Silicon Useful? Harnessing the Four Horsemen of the Coming Dark Silicon Apocalypse. In *Proceedings of the 49th Design Automation Conference*, June 2012.

[49] A. Tilli, A. Bartolini, M. Cacciari, and L. Benini. Don't Burn Your Mobile! Safe Computational Re-Sprinting via Model Predictive Control. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, 2012.

[50] A. Tucker and A. Gupta. Process Control and Scheduling Issues for Multiprogrammed Shared-Memory Multiprocessors. In *Proceedings of the 12th ACM Symposium on Operating System Principles*, 1989.

[51] S. K. Venkata, I. Ahn, D. Jeon, A. Gupta, C. Louie, S. Garcia, S. Belongie, and M. B. Taylor. SD-VBS: The San Diego Vision Benchmark Suite. In *Proceedings of the IEEE International Symposium on Workload Characterization*, Sept. 2009.

[52] G. Venkatesh, J. Sampson, N. Goulding, S. Garcia, V. Bryksin, J. Lugo-Martinez, S. Swanson, and M. B. Taylor. Conservation Cores: Reducing the Energy of Mature Computations. In *Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2010.

[53] D. Wagner and D. Schmalstieg. Making Augmented Reality Practical on Mobile Phones, Part I. *Computer Graphics and Applications, IEEE*, 29(3):12–15, 2009.

[54] L. Yan, L. Zhong, and N. Jha. User-Perceived Latency Driven Voltage Scaling for Interactive Applications. In *Proceedings of the 41st Design Automation Conference*, June 2005.

[55] B. Zalbaa, J. M. Marina, L. F. Cabezab, and H. Mehling. Review on Thermal Energy Storage with Phase Change: Materials, Heat Transfer Analysis and Applications. *Applied Thermal Engineering*, 23(3):251–283, 2003.