COMPUTATIONAL SPRINTING: EXCEEDING SUSTAINABLE POWER IN THERMALLY CONSTRAINED SYSTEMS

Arun Raghavan

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2013

Milo M. K. Martin, Associate Professor of Computer and Information Science
Supervisor of Dissertation

Val Tannen, Professor of Computer and Information Science
Graduate Group Chairperson

Dissertation Committee

André DeHon, Professor of Electrical and System Engineering

Zachary Ives, Associate Professor of Computer and Information Science

Jonathan M. Smith, Professor of Computer and Information Science

David Brooks, Professor of Computer Science, Harvard University

COMPUTATIONAL SPRINTING: EXCEEDING SUSTAINABLE POWER IN THERMALLY CONSTRAINED SYSTEMS

To my parents.

Acknowledgments

Milo Martin has been my advisor through my years in graduate school. Beyond his profound contributions to all our published research, he has influenced how I think, read, write and present my work. I hope I continue to benefit from his advise in the years ahead.

Soon after Milo and I had the first conversation about sprinting and ran through some back-of-the-envelop calculations, Tom Wenisch (Michigan) shared our impressions of initial disbelief and cautious optimism. He immediately brought on board Kevin Pipe and Marios Papaefthymiou (both from the University of Michigan) whose added expertise in thermal and electrical engineering respectively gave this project the required critical mass. Milo, Tom, Kevin and Marios were hence instrumental in the inception of this project and continue the active collaboration as of this writing.

Yixin Luo and Anuj Chandawalla performed the SPICE experiments for the many-core activation sequence under Marios' supervision at Michigan. Lei Shao put together and experimented with several phase-change heatsinks (including those used in this dissertation) and is continuing to work on thermal aspects of sprinting under Kevin's supervision at Michigan. Laurel Emurian at Penn helped put together adrenaline-2 and adrenaline-3 (the successor sprinting testbeds to adrenaline-1) and reported the first successes with sprint-and-rest for sustained workloads. Kris and SSVR read through drafts of this dissertation, catching several bugs and improving the English. Drew Hilton created the multicore x86 simulator used in Chapter 4.

Amir Roth and E. Lewis (and Milo, of course) helped make the Architecture and Compilers Group (ACG) at Penn my venue for grad school. Amir taught me CIS 501 during the first year (my first ever college level computer architecture course), hired me as teaching assistant during the next, and left me with an indelible classroom experience. Anne, Colin, Drew, Tingting and Vlad welcomed me as a fellow student into ACG. Colin helped make my initial experiences with research successful, co-authored my first couple of publications and taught me much along the way. Most of

my years in Levine 614 were spent alongside Drew and Santosh, and later Abhishek, Christian and Laurel (and sometimes Sela). They shared the SGE cluster with me before deadlines, helped debug code, and offered feedback on papers. Adam and Emily joined the ACG folks in sitting through several practice talks, patiently guiding each iteration towards a better version.

Rajeev, Sudipto and Benjamin Pierce taught excellent courses which gave me an appreciation for theory, even if I quickly learned that it wasn't my cup of tea. Instead, I was happy to collaborate with Abhishek, Jyotirmoy, Sela, Milo and Rajeev on a PLDI paper which included a lot of Greek.

I turned to Mike Felker for any departmental requirement and he responded with incredible efficiency every single time. Mark, Amy, Gail, Towanda and Lillian at the Moore Business Office facilitated equipment orders, reimbursed travels and even replaced my stolen laptop. Brittany, Charity, Cheryl, Maggie and Marissa helped with booking conference rooms, projector rentals and receiving shipments. Chip, Dan and the good folks at CETS managed the cluster, desktops and even helped with technical queries when setting up the adrenaline machines.

Doug Carmean, Mike Upton and Mark Davis hosted me as an intern with the Larrabee architecture team in Intel Hillsboro. Besides the opportunity to work on architectural exploration of a slated product, I enjoyed the conversations with several brilliant industry veterans.

Finally I would like to thank my dissertation committee—David Brooks, Andre' DeHon, Zack Ives and Jonathan Smith, who offered valuable direction after my proposal towards guiding the goals of this dissertation, and improved this document with their feedback. It has been a privilege to have them on my committee.

ABSTRACT

COMPUTATIONAL SPRINTING: EXCEEDING SUSTAINABLE POWER IN THERMALLY CONSTRAINED SYSTEMS

Arun Raghavan

Milo M. K. Martin

Although process technology trends predict that transistor sizes will continue to shrink for a few more generations, voltage scaling has stalled and thus future chips are projected to be increasingly more power hungry than previous generations. Particularly in mobile devices which are severely cooling constrained, it is estimated that the peak operation of a future chip could generate heat ten times faster than than the device can sustainably vent.

However, many mobile applications do not demand sustained performance; rather they comprise short bursts of computation in response to sporadic user activity. To improve responsiveness for such applications, this dissertation proposes *computational sprinting*, in which a system greatly exceeds sustainable power margins (by up to 10×) to provide up to a few seconds of high-performance computation when a user interacts with the device. Computational sprinting exploits the material property of *thermal capacitance* to temporarily store the excess heat generated when sprinting. After sprinting, the chip returns to sustainable power levels and dissipates the stored heat when the system is idle.

This dissertation: (i) broadly analyzes thermal, electrical, hardware, and software considerations to analyze the feasibility of engineering a system which can provide the responsiveness of a platform with 10× higher sustainable power within today's cooling constraints, (ii) leverages existing sources of thermal capacitance to demonstrate sprinting on a real system today, and (iii) identifies the energy-performance characteristics of sprinting operation to determine runtime *sprint pacing* policies.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Although transistor dimensions are projected to continue shrinking to the end of the decade [7], the accompanying growth in undesirable leakage current impedes the ability to reduce voltage. The resultant trend of scaling up the number of on-chip transistors without reducing per-transistor switching energy is a growing gap between the peak power-performance point of future chips and the sustainable cooling limits of current platforms. This sustainable cooling rate is typically specified as a thermal design power (TDP), and is commonly conflated with processor performance—the maximum operating power a chip can sustain indefinitely without overheating. In environments which preclude adoption of aggressive cooling (such as a mobile phone), the resulting observation that not all transistors on a chip can be powered all the time has led researchers in industry and academia to suggest that increasing fractions of future chips would by necessity remain "dark silicon" [7, 26, 27, 31, 47, 70, 169]. Mike Muller, the CTO of ARM, predicts over 90% of a chip to be dark silicon by the end of this decade [113].

Researchers have proposed a variety of solutions to combat dark silicon [162], from dedicating chip area for specialized, selectively utilized functional units [60] to drastically reducing voltage to near- and sub-threshold levels [44]. Modern devices already dedicate over half their chip area to specialized hardware for applications such as media codecs [34], enabling proven improvements in performance and energy efficiency (battery life) [66]. Even supposing that cramming this dedicated fraction of the chip with more specialized hardware (enabled by CMOS scaling) could yield continued benefit, thermal constraints would still limit the utilization of the remaining fraction of the chip apportioned to more general-purpose hardware (like the cores, caches, and GPUs). Other tech-

niques such as operating at reduced voltage and frequency are especially appealing in low power environments like sensor networks, but face the challenge of compensating for the lost performance of compute-intensive workloads [51, 76, 93]. In summary, the above proposals suggest partial solutions to the dark silicon phenomenon, sharing the common motif of reducing operating power to provide sustained performance for certain long running applications.

This work offers a complementary approach, observing that interactive phone applications impose an intermittent, rather than sustained computation load. The insight guiding this approach is that in such scenarios of transitory activity, it is no longer necessary to restrict operating power to sustainable heat dissipation—temporarily buffering excess heat is an alternative to the singular focus on steady-state heat dissipation prevalent in today's systems. Section 1.1 first reviews such emerging applications and proposes "computational sprinting" to meet their intermittent performance demands. Section 1.2 overviews sprinting operation, illustrating the key idea of explicitly provisioning heat buffering to extend high power sprints. Section 1.3 then outlines the goals and organization of this dissertation towards understanding and evaluating the feasibility, practicality and performance potential of computational sprinting. Section 1.4 lists the differences between this dissertation and the work published in its pursuit.

1.1 Bursty versus Sustained Computation: Re-examining Thermal Constraints

The focus on sustained performance is the right design choice for certain long-running applications like batch-mode scientific applications, or video games. However, many emerging workloads, especially phone applications, are interactive in nature; they are characterized by short bursts of intense computation punctuated by long idle periods waiting for user input [22, 180], especially in mobile settings [152].

Such mobile computing platforms—including phones and tablets— are a ubiquitous and rapidly innovating segment of the computing landscape. As users interact ever more frequently with these devices, new applications continually emerge, with ever-growing computational demands (e.g., mobile visual search [5, 6, 58, 165], highly accurate handwriting and character recognition [45, 46, 109], and augmented reality [172]). Although peak computational demands continue to rise, many applications are bursty, either because the user interacts with the application intermit-

tently (e.g., using one's phone to perform a single visual search), or because the nature of the application requires intensive computation only sporadically (e.g., recalculating a GPS route due to a wrong turn). The growing computational requirements of such bursty applications make *responsiveness*—how long a user must wait after initiating a command—a central distinguisher (alongside battery life) in the quality of a mobile platform.

Decades of research have shown that productivity and user satisfaction improve drastically as response times shrink below one second. In the early 1980's, seminal work at IBM showed that cutting transaction processing system response time from 3 seconds to 0.3 seconds doubled user productivity, reducing the user's think time between transactions by more than ten seconds [42]. In web search, even delays as short as a few hundred milliseconds measurably reduce queries per user and ad revenue [147]. To cater to these responsiveness demands within the tight energy and thermal constraints (imposed by the form-factor of existing mobile computing platforms), application developers often compromise quality—sacrificing application features and selecting sub-par algorithms—to complete compute-intensive tasks within acceptable delays [37, 58, 67].

For example, in tasks such as image segmentation (a key step in image-based search [58, 144]), responsiveness concerns have driven application designers to choose less compute-intensive algorithms that fall short of the result quality of the best known algorithms [37, 58, 67]. Advances in wireless networking have kept the most heavily-utilized smartphone application—the web browser—compute-bound and dogged by responsiveness complaints due to poor rendering performance [65, 157]. Although cloud-based computational off-loading can provide more compute resources [40, 128], such offloading does not replace demands for client-side computation, especially for applications that demand sub-second responses like user-interface updates, or applications that benefit from substantial local pre-processing like filtering and compression [58].

For applications such as the above, responsiveness may be more important than sustained performance [52]. This dissertation poses the question:

> *"What would a system look like if designed to focus on responsiveness during bursts rather than with a singular focus on sustained performance?"*

Computational Sprinting. The central hypothesis of this dissertation is that platforms should be engineered to run beyond thermal design power to provide bursts of intense computational performance during moments of application demand. Similar to human sprinters who can maintain peak

speeds only for short durations, such *computational sprinting* (for example, by activating additional otherwise inactive "dark" cores or boosting frequency) is unsustainable; because the heat generated when sprinting cannot all be dissipated, continued sprinting would eventually push the chip towards an unsafely high temperature. However, the time lag before reaching critical temperatures—a result of every material's ability to absorb heat, termed *thermal capacitance*—is an opportunity to boost performance significantly. By exploiting thermal capacitance to temporarily buffer excess heat during high-power bursts of computation and dissipating the stored heat during extended idle periods (for example, when a phone rests in a user's pocket), computational sprinting conceptually "borrows" energy from periods of inactivity to exceed sustainable power in response to intermittent application activity. This dissertation evaluates this hypothesis using both simulation (on analytical models) and a hardware-software implementation of sprinting on a real system with induced thermal constraints.

Proposed advances to the state-of-the-art. Shortly after the inception of this project, commercially available processors such as Intel's Sandy Bridge began to exploit the headroom from thermal capacitance to dynamically scale frequency and voltage [141]. This embodiment of sprinting is currently restricted to modest peak-performance boosts (25% over tens of seconds), limited firstly by the available thermal capacitance in existing systems and secondly by the relative energy-inefficiency of voltage-frequency boosting (which incurs a super-linear power increase to provide linear increase in peak-speedup). Given the projections of a much higher gap between peak and sustainable power (10×—see Chapter 2) and the near-second response time requirements of interactive applications, this dissertation instead proposes: (i) explicitly engineering systems to provide the performance of a system with 10× thermal design power for brief durations—short enough that the system does not overheat, yet sufficient to enhance the responsiveness of interactive applications by up to 10× and (ii) capitalizing on the energy efficiency of parallelism to sprint by activating tens of otherwise idle cores in response to application activity.

1.2 Computational Sprinting Overview

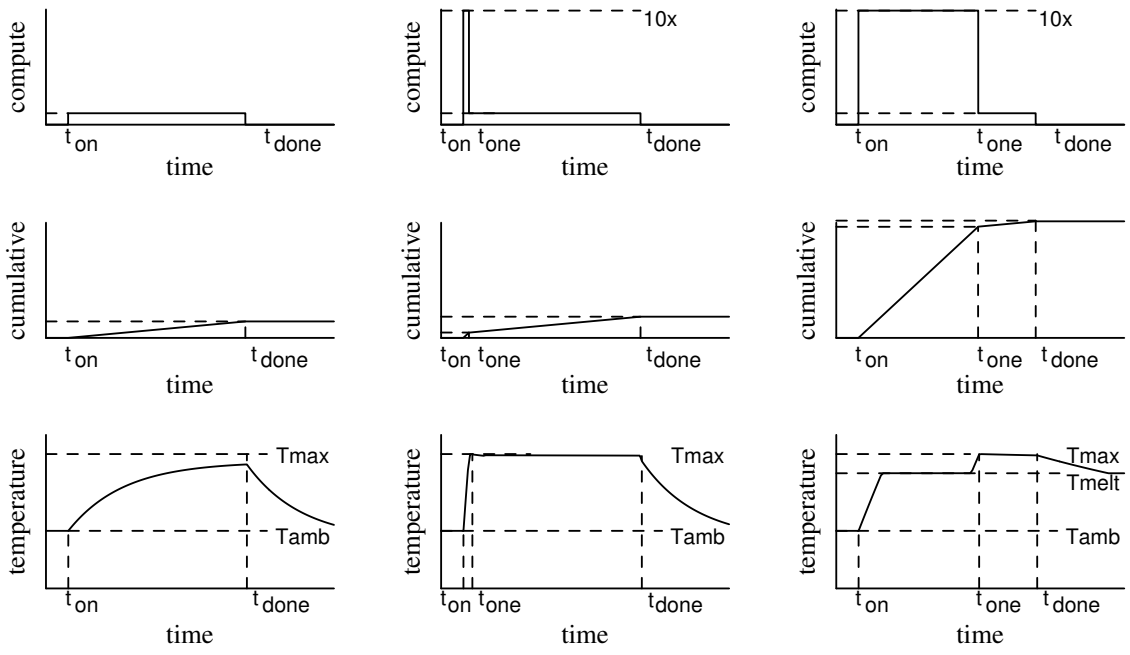
This section overviews the operation of computational sprinting, considering the example of *parallel* sprinting by computing with ten cores on a platform which can sustain just a single core. The operating principle illustrated below applies also to sprinting by boosting frequency, although the

amount of computation performed during such a sprint may vary. The following discussion foreshadows some of the thermal aspects that enable sprinting; a more detailed exposition follows in Section 4.4.

Sustained operation. Figure 1.1a shows the thermal response of a conventional system starting from idle. The core becomes active at time t_{on}, causing the temperature of the system to rise asymptotically toward T_{max}. The system computes until time t_{done}, at which time the core becomes idle and the temperature begins to fall asymptotically toward $T_{ambient}$. For a given $T_{ambient}$ and T_{max}, the maximum sustainable thermal power is determined by total thermal resistance of the package and heatsink (but is independent of thermal capacitance). In contrast, the rate at which the temperature rises (and falls) is determined by both thermal resistance and capacitance.

Sprinting operation. Sprinting mode operation is shown in Figure 1.1b. In this example, the system is initially idle and the system temperature matches that of the ambient environment $T_{ambient}$. At time t_{on}, an external event (*e.g.*, user input) triggers demand for a burst of computation, and it initiates a parallel sprint by activating all cores. As the heat generation is greater, the chip temperature rises faster than in sustained operation. In this example, the temperature reaches T_{max}, and thus the sprint duration exceeds the maximum sprint duration of the system. When the temperature reaches the maximum permissible value at time t_{one}, the system terminates the sprint by disabling all but one core; remaining work is completed by this core. During this interval (from time t_{one} to t_{done}), because the thermal system is designed to sustain the operation of a single core, temperature remains stable near T_{max}. When the computation is done (t_{done}), all cores become idle, hence the system begins to cool.

Augmented sprinting operation. Because today's systems (for example phones) are not designed with a focus on thermal capacitance, the available buffer from existing sources of thermal mass (such as the silicon die) can severely limit the duration of sprinting. To extend sprinting with 10× the sustainable power for up to one second, one approach is to increase the thermal capacitance of the system by placing a block of phase change material close to the die (described in Chapter 4). This material adds to the thermal capacitance by its *specific heat* (the amount of energy to change the temperature of a gram of the material by one degree) and more importantly its *latent heat of fusion* (the amount of energy to melt a gram of the material). Figure 1.1c shows the same computation in sprint mode on such an augmented system. The temperature rises as before, but when the temperature reaches the melting point of the material (T_{melt}), the extra thermal energy injected into the



(a) Sustained execution (b) Sprint execution (c) Augmented sprint

Figure 1.1: **Sprinting operation.** Cores active (top row), cumulative computation (middle row) and temperature (bottom row) over time for three execution modes: (a) sustained, (b) sprint, and (c) sprint augmented with phase change material.

system is absorbed by the melting process, allowing the system to continue to execute for a period *without* increasing the temperature. Only once the material has fully melted does the temperature begin to rise again. Similarly, when the system cools, its temperature is constant as the material returns to its solid phase. Overall, in this example the additional thermal capacitance allows the system to perform significantly more computation during the sprint interval, which can lead to an order-of-magnitude reduction in time-to-completion for a computational task.

As shown by this example increased capacitance is paramount — it sets the upper bound on the total energy expended during a sprint. However, thermal resistance between the thermal capacitor and the die bounds the intensity of the sprint (*i.e.*, number of cores activated). Section 4.4 analyzes the thermal implications of the different components of a mobile device. The key observation is that the thermal capacitance provides the temporary window for computational sprinting. Other startup transients — electrical, architectural, and software, should be small fractions of this window to obtain significant responsiveness gains.

1.3 Dissertation Structure and Goals

This dissertation aims to substantiate the thesis that explicitly engineering computing systems to intensely sprint beyond sustainable power can enable large responsiveness benefits for bursty applications.

The next two chapters provide the background for this work: Chapter 2 identifies the limits of sustainable performance due to process technology trends which motivate reexamining sustainable power constraints. Chapter 3 describes key power-temperature relationships which provoke the insight that thermal capacitance can temporarily buffer unsustainable compute power. The remainder of this dissertation is then organized around three goals: (i) analyzing the feasibility of engineering a system which can provide the responsiveness of a platform with 10× higher TDP within today's cooling constraints, (ii) empirically confirming that the principle of leveraging thermal capacitance (from both specific heat and latent heat) indeed enables intense sprinting on a real system, and (iii) identifying the energy-performance characteristics of sprinting operation which influence operating decisions such as *how* to sprint.

Contribution #1: Analyzing the Feasibility of Engineering a Future Sprinting System. Today's systems are not designed with the goal of exceeding sustainable power by a factor of 10×. Whereas sprinting requires sufficient thermal capacitance to buffer this heat, existing thermal design guidelines overlook thermal capacitance and chiefly consider thermal resistance between the die and its ambient environment (the key factor behind TDP) [8]. Similarly, conventional mobile phone batteries and power distribution networks are engineered to supply at most a few amps, raising challenges in supplying high load currents when sprinting. The inherently unsustainable nature of sprinting necessitates light-weight thermal monitoring and power control mechanisms in hardware and/or software to enable safe functioning without compromising performance. In contrast, the heavy-duty thermal throttling in today's systems are intended as (seldom invoked) responses to thermal alarm trips—situations where performance is not the primary objective. Thus, the first goal of this dissertation is to investigate the feasibility of sprinting, broadly considering immediate thermal, electrical and hardware/software challenges. Chapter 4 addresses this goal.

Contribution #2: Demonstrating the Practicality of Sprinting on a Real System. To progress beyond a measure of feasibility gathered from modeling and simulation, the second goal of this dissertation is to empirically demonstrate sprinting on a real system. Despite being motivated

by future trends, the basic faculties for sprinting are present even in today's systems: (i) thermal capacitance is a material property inherent in all objects, including processor packages, and (ii) most modern chips support multiple power modes for energy savings. In a modern desktop chip, the range of power-performance configurations are separated by tens of watts and a large performance difference based on frequency and active core settings. Chapter 5 evaluates the practicality of sprinting on a real system today, leveraging thermal capacitance from: (i) the heat spreader internal to the processor package (specific heat), and (ii) a paraffin-based phase-change material mounted on the package (latent heat).

Contribution #3: Identifying the Energy-Performance Properties of Sprinting. After evaluating the premise of utilizing thermal headroom to sprint, the third goal of this dissertation is to identify and evaluate the responsiveness and energy impact of sprinting. Sprinting capitalizes on thermal capacitance, which is essentially a finite energy resource—a total amount of heat which can be absorbed before the die gets too hot. As a computational resource, this energy can be used either for short, high-power sprints or alternatively for lower power sprints which can last longer. Additionally, performance does not always scale linearly with power, suggesting trade-offs based on *how* a system sprints—using parallelism alone is expected to perform more computation than an alternative which uses the same total energy when boosting voltage and frequency. Further, in an operating regime with bursts of computation separated by large periods where the entire system is idle, the energy saved by turning off background components sooner can be significant (such as the screen on a phone/tablet or a large shared cache on-chip which contributes to processor energy). This dissertation describes the relationship between the components of system power (idle, compute, and background power) and performance to understand when and how sprinting can benefit application responsiveness and even save energy. Chapter 6 analyzes the energy-performance properties of sprinting.

This dissertation primarily focuses on the thermal and energy-performance aspects of sprinting and qualitatively discusses practical considerations such as power supply and delivery, reliability and cost.

1.4 Differences from Previously Published Versions of this Work

Chapter 4 expands on the published versions of the approach and feasibility study [137, 138] by examining the limitations of sprinting on today's mobile systems, providing further exposition on emerging battery and hybrid ultracapacitor power supplies, discussing the thermal properties of phase-change materials, and analyzing the runtime and dynamic energy breakdown of the simulated workloads. Chapter 5 characterizes the thermal response of the testbed and its ability to sprint using various core-frequency settings beyond prior published work [135]. Chapter 6 additionally characterizes the power, performance and energy of the different modes of sprinting, explicitly compares experimental results with modeled estimates [136], and proposes and evaluates a gradual sprint-pacing policy which has not previously been published (Section 6.5.3).

Chapter 2

Background on Dark Silicon

Researchers across academia and industry have raised the need to revisit how on-chip transistors are utilized in light of recent technology scaling trends. For almost four decades since the observation of Moore's law [121], several generations of transistor shrinking caused commensurate reduction in both switching energy and gate delay (hence increase in frequency) [41], fueling exponential growth in peak-performance without increasing power consumption. However, as transistor sizes approach physical limits—for example, today's gate-oxide thickness of 1-2 nm is approximately five silicon atoms wide—these "rules of thumb" have begun to break down.

This chapter summarizes the expected increase in power density based on published industry and academic estimates. As increased power density translates to excess heat, these estimates also predict the fraction of chip activity that can be sustained under fixed cooling constraints, with the remainder of the chip contributing to *dark silicon*. Mike Muller (CTO of ARM) has spoken publicly about the dark silicon problem, predicting that by 2019 only 9% of the transistors on a chip can be active at any time [113]. This chapter then overviews previously suggested approaches to mitigate dark silicon and observes that some of these techniques are already deployed in current mobile devices. Based on this background, this dissertation proposes computational sprinting as an alternative approach to utilize dark silicon without being constrained by thermal design power for applications that do not demand sustained computation.

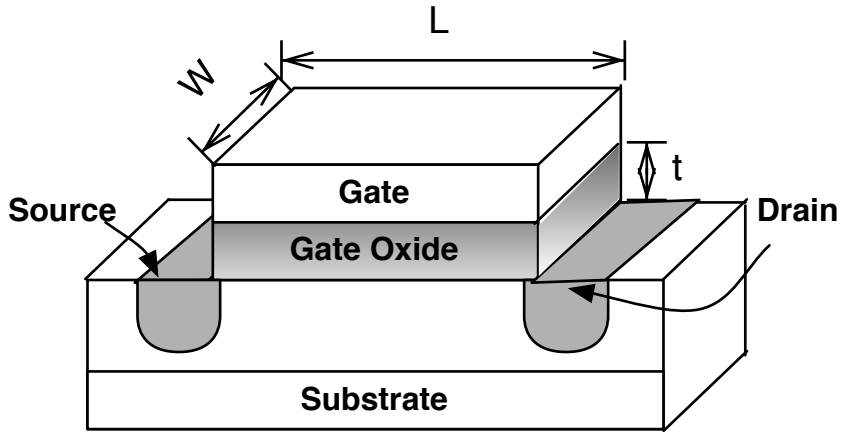


Figure 2.1: A basic MOS gate

2.1 Basic Transistor Operation and Dennard Scaling

The simplified overview of MOS (metal oxide semiconductor) operation provides a basic framework to understand why transistor scaling has changed expectations of improvements in processor power and performance. Several textbooks and papers have been written about the topics relating to static and active power in CMOS technologies and their relevance to processor design. The Synthesis Lectures on "Computer Architecture Techniques for Power-Efficiency" present a detailed summary [85].

Figure 2.1 shows the basic model of a MOS gate with its four terminals (source, drain, gate and substrate). The gate is isolated from the rest of the device by an oxide layer that acts as an insulator (dielectric). The device is constructed by doping the source and drain to contain particles of opposite polarity as the substrate; therefore in its idle/off state, the source and drain are cut off from each other by the region under the gate-oxide (called *depletion region* due to its lack of the right type of carrying particles). Applying a voltage to the gate creates an electric field across the gate oxide and attracts charge carriers into the region under the gate-oxide (charging the dielectric capacitance across the gate oxide). When the gate voltage is raised above a certain threshold (V_{th}), sufficient particles accumulate below the gate to form a *channel* between the drain and source. In this active/on state, when a voltage (V) is applied between the source and the drain, it causes an electric field across the channel inducing a flow of current.

The active power consumption of the transistor results from the charging and discharging of gate-capacitance as transistors switch when gate voltage is toggled relative to V_{th}. For a CMOS based processor, the active power is modeled as:

$$P_{active} = \alpha \cdot C \cdot V^2 \cdot \ f \tag{2.1}$$

where each of factor depends on the transistor dimensions (length L, width W and gate-oxide thickness t) as follows:

- Activity factor (α) represents the number of transistors that are switching; when the size of a single transistor decreases, more such transistors can fit into a chip of a given size. Area $A = LxW$, $\alpha \propto 1/A$.
- Gate capacitance (C) represents the charge that can be stored due to a potential drop across the gate and depends on the thickness of the gate oxide and the area of the transistor. $C \propto t/A$.
- Threshold voltage (V_{th}) and supply voltage (V) are used to induce electric field for carrier motion in MOS transistors. Because electric field varies with potential *gradient*, when transistor size is reduced, voltage can be reduced by the same fraction to maintain the same field across the smaller length. $V_{th} \propto t$, and $V \propto L$.
- Maximum operating frequency (f_{max}) represents the switching speed of a transistor which diminishes with the delay (d) for charging and discharging the gate capacitance. Because the total charge is $C \cdot V$, the delay depends on the *rate* of charge/discharge—or the amount of current. This drain current I has previously been modeled in the literature as $I \propto W/(L \cdot t) \cdot (V - V_{th})^2$ [41]. Therefore, $d \propto C \cdot V/I$ and $f \propto 1/d$.

With ideal scaling (also called Dennard scaling after its first proponent [41]), when the length and the width, as well as gate-oxide thickness are scaled by a factor of κ (to L/κ, W/κ and t/κ respectively), the threshold voltage scales to V_{th}/κ, the supply voltage to V/κ, and capacitance to C/κ. As a result, a chip of the same area can contain κ^2 more transistors each switching at a frequency of $f \cdot \kappa$ while still maintaining the same active power [41] (Table 2.1). Hence, Dennard scaling implies exponential performance gains at constant area and constant power just by shrinking transistor sizes.

Parameter	Description	Derivation	Dennard scaled value (by κ)
L	Transistor length	Process technology	L/κ
W	Transistor width	Process technology	W/κ
t	Gate-oxide thickness	Process technology	t/κ
A	Transistor area	$A = L \cdot W$	A/κ^2
α_{max}	Maximum activity factor	$\alpha_{max} \propto chip\ area/A$	$\alpha_{max} \cdot \kappa^2$
C	Gate capacitance	$C \propto t/A$	C/κ
V_{th}	Threshold voltage	$V_{th} \propto t$	V_{th}/κ
V	Supply voltage	$V \propto L, W$	V/κ
I	Drain current	$I \propto W/(L \cdot t) \cdot (V - V_{th})^2$	I/κ
d	Charging/discharging delay	$d \propto C \cdot V/I$	d/κ
f_{max}	Max. operating frequency	$f_{max} \propto 1/d$	$f_{max} \cdot \kappa$
P_{active}	Active power	$P_{active} = \alpha \cdot C \cdot V^2 \cdot f$	P_{active}/κ^2
power density	Active power per unit area	P_{active}/A	power density × 1

Table 2.1: Structural and operational parameters of MOS transistors with ideal (Dennard) scaling factor κ.

2.2 Static Power and the Limits of Dennard Scaling

Unfortunately, Dennard scaling has stalled. Until transistor sizes fell below a tenth of a micrometer, active power dominated static power and Dennard's scaling trends remained prophetic. However, with continued shrinking beyond the 90nm technology node, the growing proximity of terminals causes increasing amounts of current to leak through unintended paths [25]. The contribution of static power due to leakage current is approximately 30% of total chip power today.

Static power is modeled as:

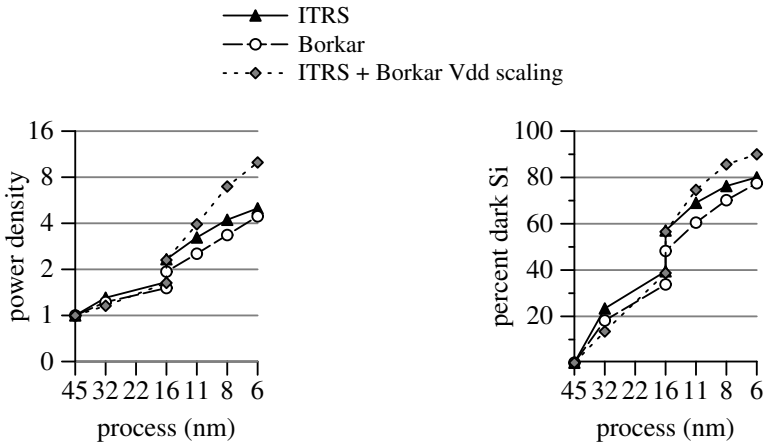
$$P_{static} = V \cdot I_{leak}$$

Although current leaks along several different paths (six types of leakage current are commonly classified [85, 142]), leakage power in modern processors is chiefly attributed to two sources [32, 87, 122]: (i) from the source to the drain when the device is off, *i.e.*, when the gate is held below V_{th} (called sub-threshold leakage), and (ii) across the gate-oxide. The increasing trend in these leakage currents precludes further scaling of threshold voltage. As seen above, the frequency of the device depends on $(V - V_{th})$, hence the supply voltage can also not be reduced without compromising performance.

In addition, some predictions estimate that the scaling in capacitance will also not keep pace with device density scaling. For example, Borkar *et al.* [27] predict a 25% capacitance reduction

Parameter	Estimate source	45nm	32nm	22nm	16nm	11nm	8nm	6nm
Device density	ITRS	1.00	2.00	4.00	4.00	8.00	16.00	32.00
	Borkar	1.00	1.75	3.06	5.36	9.38	16.41	28.72
Capacitance	ITRS	1.00	0.69	0.48	0.35	0.22	0.14	0.09
	Borkar	1.00	0.75	0.56	0.42	0.32	0.24	0.18
Voltage	ITRS	1.00	0.93	0.84	0.84	0.75	0.68	0.64
	Borkar	1.00	0.90	0.83	0.79	0.77	0.76	0.75
Frequency	ITRS	1.00	1.09	1.22	2.38	3.21	4.17	4.38
	Borkar	1.00	1.15	1.26	1.37	1.43	1.49	1.54

Table 2.2: **Scaling estimates.** Data from Borkar [26, 27, 47], ITRS [7].



(a) Power density

(b) Percent dark silicon

Figure 2.2: **Power density and dark silicon trends.** Data from Borkar [26, 27, 47], ITRS [7], and ITRS with Borkar's more pessimistic voltage scaling assumptions [26].

versus 75% density (transistors per unit area) increase per process generation. In summary, the result of these scaling limitations in voltage and capacitance is that active power grows with every process generation. Because increasing power translates to higher power density for a chip of the same size, the combined scaling trends now point to chips that run hotter with each process generation, even when frequency remains flat. Attempts to increase frequency further exacerbate this effect.

2.3 Thermal Design Power and Dark Silicon

An increasing chorus of academic and industry veterans have issued similar-spirited, albeit quantitatively varying warnings, about the increasing power density in current and future processors based

on their estimates of slowing voltage scaling [7, 26, 27, 47, 70, 113, 169]. Table 2.2 shows two such representative estimates. The International Technology Roadmap for Semiconductors (ITRS) estimates are optimistic in terms of supply voltage scaling [7]. In contrast, Borkar [27] expects voltage to scale less aggressively based on Intel's roadmap, and is likely more realistic, but does not scale frequency to the higher supply voltage. Therefore, a third trend that combines the ITRS estimates with Borkar's more conservative voltage scaling is an interesting point to consider for peak performance.

Figure 2.2a shows the power density for a chip of the same size for the above three estimates of process scaling. All three trends predict power density to increase to over 6× beyond the 8nm process node. If the devices are operated at peak frequency (ITRS + Borkar scaling), the power density reaches 10× that of a similar sized chip in 45nm technology. Until recently, the challenge of increased power density was passed along to package designers by specifying higher thermal design power (TDP) for chips, resulting in CPUs and GPUs that dissipate 100+ Watts. However, both high-end chips and mobile chips are reaching the thermal limits of active and passive cooling, respectively. Without innovation, either physical chip size must decrease each generation (thereby ending the decades of benefit the industry has reaped from Moore's Law) or the active fraction of the chip must be decreased. Figure 2.2b shows that up to 90% of a chip may by precluded from sustained utilization under the same cooling constraints as the 45nm generation. Mike Muller (CTO of ARM) has spoken publicly about the dark silicon problem since at least 2009, predicting that by 2019 only 9% of the transistors on a chip can be active at any one time [113].

2.4 Approaches to Mitigate Dark Silicon

Taylor [162] introduces a taxonomy of the "four horsemen" to classify suggested proposals to combat dark silicon. These approaches have largely focused on reducing thermal design power power by decreasing one of its contributing factors from Equation 2.1—frequency/voltage or active fraction.

Reducing die size. One option—using miniaturization to reduce die area and derive incidental savings in manufacturing cost—forsakes the performance improvements that have driven the industry forward. In fact, even the purported cost and area savings are questionable: die size is but one component of a chip, and it is unclear if other packaging constraints, notably the C4 bump pitch

required to carry supply power into the chip, would allow reduction in the area and cost of the entire chip [70, 162].

Reducing activity factor. Another approach involves reducing activity factor by dedicating fractions of die area to accelerators and special function units [13, 48, 60, 169, 170]. Selectively activating these fractions of the chip to efficiently execute specific types of workload reduces power consumption. For example, special function H.264 codecs are used for media playback [66], during which time other components of the chip are usually idle. Current processors already employ several such special function units. In addition, a large part of a chip's area is dedicated to "uncore" parts of the processor such as caches and interconnects which usually see much less activity than the processor "core." For example, on the testbed system used in this dissertation, doubling frequency increases uncore power by only 30%, whereas core power increases by 200% (Table 6.2).

Reducing voltage and frequency. A third proposal reduces active power by lowering voltage to near-threshold levels, operating the device at low frequency so that energy is minimized at the cost of large performance degradations [44]. To reclaim some of the lost performance, these approaches look to exploit either SIMD-like data parallelism or increase the number of cores [51, 76, 93]. Although appealing in the context of extremely low-power environments like sensor nodes, the performance trade-offs and susceptibility to process variability are significant challenges to general-purpose deployment.

2.5 Dark Silicon in Mobile Chips

The dark silicon trend is perhaps most acute for emerging smart phone and tablet devices, which are already exhibiting several symptoms of energy limitations and significant inactive (or "dark") silicon.

First, the per-area power densities are lower for phone/tablet chips than desktop/server chips. The chips used in phones and tablets comprise multiple CPU cores at 1 GHz, powerful GPUs, and special-purpose accelerators for media computations. Although limited to just a few watts due to both battery life and thermal constraints, these chips have substantial die area (Apple's A4 is 53 mm^2, Apple's A5 is 122 mm^2, and NVIDIA's Tegra 2 is 49 mm^2). In comparison, Intel's "Sandy Bridge" chips range from 149 mm^2 for a dual-core chip (216 mm^2 for a quad-core chip) with a

TDP ranging from 17 W at 1.8 GHz to 65 W at 3.3 Ghz. Thus, although the phone/tablet chips have only 3× less area, their TDP is 7× to 25× lower.

Second, the power density of the individual cores (including caches) is higher than the overall chip. For example, the die photo of the Apple A5 chip indicates that each ARM Cortex A9 core (including L1 and L2 caches) is approximately 7% of the total die area (or less than 9mm^2 in 45 nm); the dual-core GPU plus both A9 cores take up less than half of the die area. Filling the entire A5 die with 0.5W A9 cores would result in a power draw of over 7 W, which is at least 2× or 3× more than what the A5 sustains based on battery life calculations.

Third, mobile chips have already adopted many of the commonly suggested approaches for mitigating the dark silicon problem. For example, the A5 die photo shows that less than half the die area is used for the CPUs and GPU; much of the rest of the die is used for heterogeneous cores and custom accelerators (*e.g.*, for energy-efficient multimedia playback). In addition, mobile chips employ sophisticated active power management and adaptive throttling to dynamically limit maximum power dissipation (*i.e.*, the various CPUs, GPUs, and special-purpose units may not be concurrently activated without exceeding TDP).

2.6 Chapter Summary

Process technology trends indicate that future chips could be thermally constrained to sustain as little as 10% of their peak performance. Current proposals to derive performance in the presence of increasingly dark silicon seek to reduce active power by lowering chip activity—either by selectively powering dedicated special function units, or reducing voltage and frequency. Although such approaches partially address the dark silicon phenomenon to benefit certain classes of applications, there remains significant opportunity to harness the peak performance potential enabled by Moore's law. The next chapter overviews key power-temperature relationships to motivate the approach proposed in this dissertation,

Chapter 3

Background on Thermal Response

The insight behind computational sprinting is that sustainable power can be exceeded for short durations of time before any temperature violations may occur. The average operating power for long running operations is restricted by the heat-venting ability of the computing system; to reach a sustainable equilibrium where the system never overheats, the heat generated during operation must not exceed the heat dissipation rate of the cooling solution. Section 3.1 first describes the focus of such conventional cooling specifications, in which the processor settles at a steady-state equilibrium temperature determined by its operating power and the system's *thermal conductivity*, *i.e.*, the rate at which it dissipates heat to the ambient environment.

However, it is possible to operate at higher power for short bursts, provided the excess heat can be temporarily stored (*i.e.*, without violating temperature margins). Section 3.2 describes how heat storage resulting from *thermal capacitance* influences the rate of temperature increase. Section 3.3 explains how a second type of heat storage, *i.e.*, latent heat of phase change, can further increase the time it takes for a material to heat or cool. Subsequent chapters utilize such sources of thermal capacitance for sprinting.

A system's capability to sprint—the extent to which it can exceed sustainable power and the duration for which it can do so—depends on the thermal resistance and capacitance present between the processor and surrounding components. Section 3.4 describes how a computing platform modeled as a network of thermal resistances and capacitances can estimate its thermal response to input power. Subsequent chapters use such models to study the feasibility of sprinting and to explain the observed thermal response in the sprinting testbed. However, this work is not the first to sug-

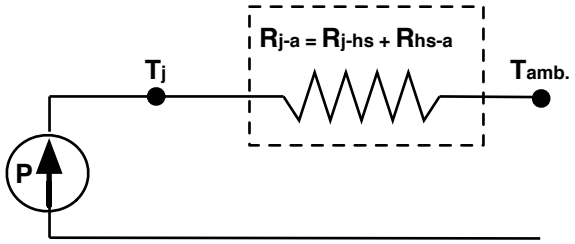


Figure 3.1: Steady state thermal equivalent model.

gest thermal-aware processor operation. Section 3.5 overviews prior work which use temperature estimates to detect thermal emergencies, prevent hotspots, and enhance reliability.

3.1 Sustainable Operation at Thermal Design Power: Role of Thermal Resistance

In conventional systems, the cooling solution is specified such that the die temperature always remains below a rated maximum value when the processor operates at its expected peak power. The temperature of the processor is measured at a single point called the junction (T_j). When the processor is active, it produces heat at the rate of P watts, causing the junction temperature to increase with respect to its neighboring components. The ambient temperature (T_{amb}) reflects the temperature of the surrounding environment (such as room temperature) and is thus relatively constant. The temperature gradient ($T_j - T_{amb}$) causes heat to flow from the processor to the ambient environment.

3.1.1 Thermal Resistance

Figure 3.1 illustrates this steady-state heat dissipation path. The rate of heat flow is limited by the aggregate thermal resistance along the path ($R_{j \cdot a}$). For example, in a system with a heatsink mounted on the processor, the thermal resistance would be the sum of thermal resistance from the processor to the heatsink via conduction ($R_{j \cdot hs}$) and from heatsink to air via passive (still air) or forced convection with a fan ($R_{hs \cdot a}$). Thermal resistance, expressed in $^\circ C$/watt, is analogous to electrical resistance—a temperature differential of $\Delta T\ ^\circ C$ across a thermal resistance of $R\ ^\circ C/W$ causes an instantaneous heat flow of $\Delta T/R$ watts. Conversely, for a given cooling solution with

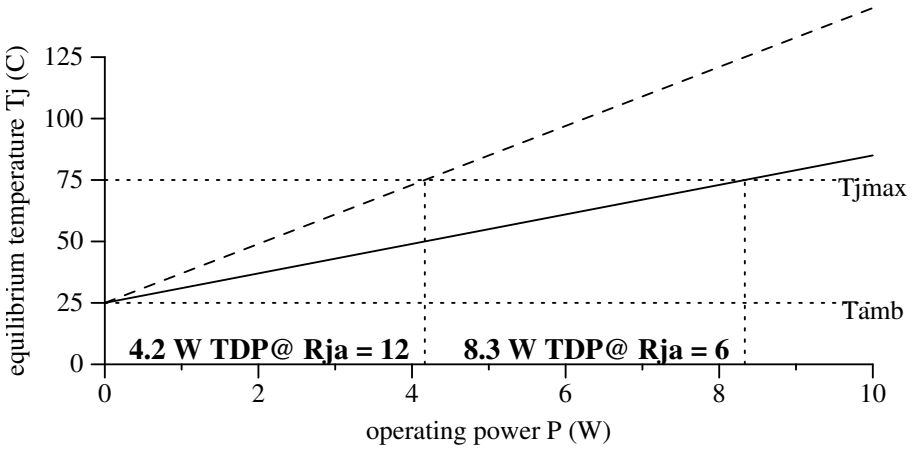


Figure 3.2: Junction temperature vs. operating power for $R_{j\cdot a}$ of (i) 6 K/W (solid line) and (ii) 12 K/W (dashed line) . Thermal design power is the maximum power at which the junction temperature remains at or below T_{jmax}.

---

thermal resistance $R_{j\cdot a}$, sustained operation at P watts will cause an equilibrium junction temperature based on the rate of heat generation and the rate of heat of dissipation:

$$\begin{aligned} Rate\ of\ heat\ generation &= Rate\ of\ heat\ dissipation \\ \implies P &= \frac{T_j - T_{amb}}{R_{j\cdot a}} \\ \implies T_j &= T_{amb} + P \cdot R_{j\cdot a} \end{aligned} \tag{3.1}$$

3.1.2 Thermal Design Power (TDP)

The processor manufacturer specifies the maximum temperature T_{jmax} that the die can sustain without causing it to overheat. For a given cooling solution (*i.e.*, a given $R_{j\cdot a}$ value), the rate of heat dissipation ($(T_{jmax} - T_{amb})/R_{j\cdot a}$) is therefore the maximum operating power that the system can sustain (thermal design power or TDP). Exceeding TDP will cause the junction temperature to settle at a value higher than T_{jmax} and can hence cannot be sustained.

Figure 3.2 shows the above dependence of junction temperature on input power for an example system with $T_{jmax} = 75°C$ and an ambient temperature $T_{amb} = 25°C$ for two different cooling solutions. With $R_{j\cdot a} = 6°C/W$, the maximum sustainable power is 8.3 W. For a less aggressive

cooling solution that can only dissipate heat at half the rate ($R_{j \cdot a} = 12°C/W$), the sustainable power is correspondingly also reduced to half (4.2 W). Hence, using this conventional approach, the processor is typically constrained to operating within the thermal design power determined by the cooling solution of the system.

3.2 Transient Temperature: The Role of Thermal Capacitance

Although a system attains an equilibrium temperature whose value is determined by thermal resistance, it does not reach this temperature instantaneously. When heat is applied to an object such as a metal, or a die made of silicon, its temperature increases over time at a rate defined by the object's thermal capacitance. Thermal capacitance thus offers an insight for an alternative to always operating within thermal design power: it is possible to exceed TDP temporarily without causing overheating. The below derivation calculates the transient temperature response to input power when considering thermal capacitance.

3.2.1 Thermal capacitance

Thermal capacitance is defined as the amount of heat required to increase an object's temperature by $1°C$ (or 1 K), and is expressed as the product of a material's specific heat and mass. For an input power P, the additional heat δQ built up over a time interval δt is:

$$\delta Q = P \cdot \delta t$$

By definition, for an object with thermal capacitance C J/K, the temperature increase (δT) resulting from applying Q joules of heat can therefore be given as follows:

$$\begin{aligned} C &= \frac{\delta Q}{\delta T} = \frac{P \cdot \delta t}{\delta T} \\ \implies \delta T &= \frac{P \cdot \delta t}{C} \end{aligned} \tag{3.2}$$

3.2.2 Transient thermal analysis

Figure 3.3 shows the thermal equivalent network including the thermal capacitance of the processor (C). In the context of processor execution, if P is the operating power and P_{out} is the rate of heat

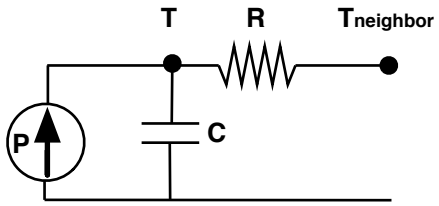


Figure 3.3: 1-stage R-C thermal network with source of input power P.

---

dissipation, then the difference between input and output power ($P - P_{out}$) causes the temperature of the processor to increase. From Equation 3.2:

$$\delta T = \frac{(P - P_{out}) \cdot \delta t}{C} \tag{3.3}$$

P_{out} is the rate of heat flow as seen earlier in Equation 3.1:

$$P_{out} = \frac{T - T_{neighbor}}{R}$$

Hence, Equation 3.3 can be rewritten as:

$$\delta T = \frac{\delta t}{R \cdot C} \cdot (P \cdot R - T + T_{neighbor}) \tag{3.4}$$

The above equation can be solved in the closed form to derive the thermal response of the system:

$$\int \frac{dT}{P \cdot R - T + T_{neighbor}} = \int \frac{dt}{R \cdot C}$$

$$\implies T = T_{neighbor} + P \cdot R - k \cdot e^{-\frac{t}{R \cdot C}} \tag{3.5}$$

The constant of integration k can be determined from the initial conditions of the system when the temperature of the processor at time $t = 0$ is $T(0)$. Thus, the transient temperature of the processor is given as:

$$T(t) = T_{neighbor} + P \cdot R - (T_{neighbor} + P \cdot R - T(0))e^{-\frac{t}{R \cdot C}} \tag{3.6}$$

How does thermal capacitance affect temperature? Figure 3.4 shows examples of transient thermal response for two systems with different thermal capacitance (C = 0.1 J/K and C = 1.0 J/K), in response to varying intensities of operation (1 W and 2 W). In both cases, the thermal resistance to the ambient is fixed at 50 $°C/W$ and the system is initially idle with the temperature at time

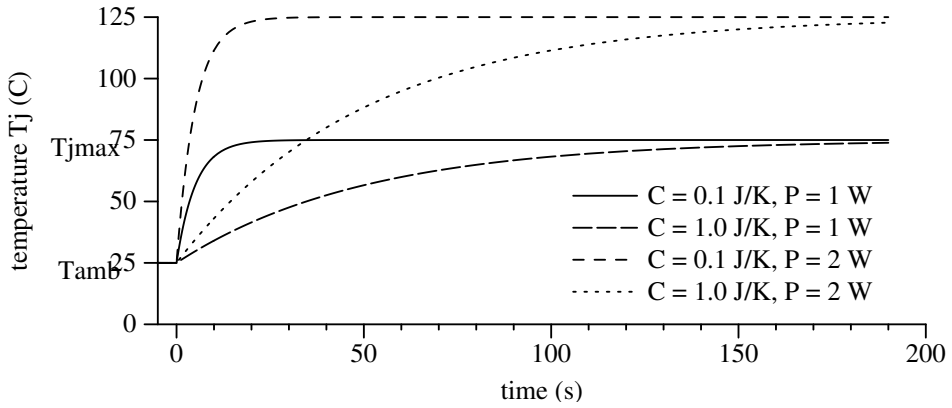


Figure 3.4: Junction temperature increasing over time with varying thermal capacitance and input power, for fixed thermal resistance to ambient 50 K/W . Initially, the system powered off (input power P = 0 W) and is at equilibrium with $T_j = T_{amb}$. At time t = 0, power is turned on causing the temperature to rise as a function of power and thermal capacitance.

$t = 0$ being T_{amb}. Hence, for a maximum rated temperature of $T_{jmax} = 75°C$, the TDP of both systems is $(75°C - 25°C)/50\ °C/W = 1\ W$. Thus, for an operating power of 1 W, both systems asymptotically approach T_{jmax}. However, with larger thermal capacitance—0.1 J/K vs 1.0 J/K, the temperature rises visibly more slowly. At 2× the operating power (2 W), the final equilibrium temperature approaches 125 $°C$ ($T_{amb} + P \cdot R$), which exceeds the junction rating of $75°C$. With a capacitance of 0.1 J/K, the system would overheat in ≈3s; however, with thermal capacitance of 1.0 J/K, it takes ≈30s before the temperature exceeds T_{jmax}. Increasing thermal capacitance therefore results in an increase in rise time for fixed thermal resistance and power.

How does power affect transient temperature? Comparing the curves with the same thermal capacitance in Figure 3.4, the temperature rises more steeply with higher power. For example, thermal capacitance of 1.0 J/K, with 1 W operating power, temperature reaches 50 $°C$ in ≈34 s, whereas with twice the power, it takes approximately half as long to reach the same temperature. The same effect is observed with lower thermal capacitance of 0.1 J/K as well, although the absolute time it takes to reach the same temperature ($50°C$) is now lower (3.5 s and 1.6 s at 1 W and 2 W respectively).

Cool down time. Equation 3.6 can be used to also understand how the system cools down after input power is turned off. For a processor that has been operating at TDP, the initial equilibrium temperature would be T_{jmax} (again fixed at $75°C$). Hence, reducing operating power to $P = 0$ at

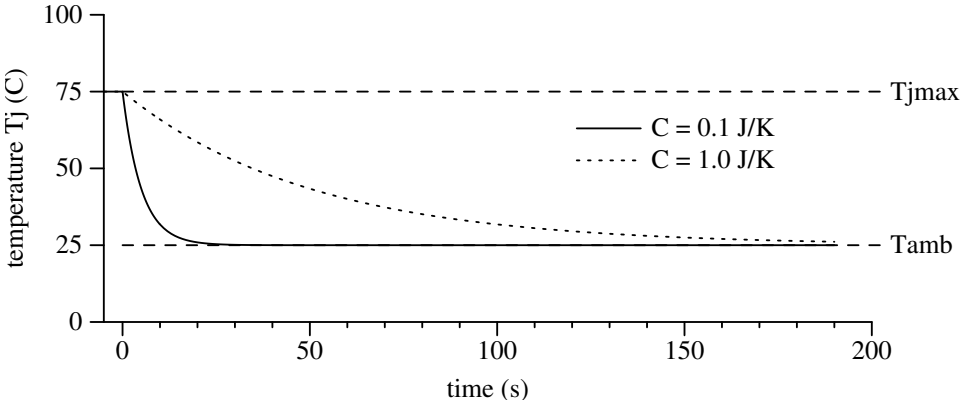


Figure 3.5: Temperature falling transient. When the system is turned off ($P = 0$) from an initial temperature of T_{jmax} (at time t = 0), its temperature decreases asymptotically towards T_{amb}. The cool down interval increases with larger thermal capacitance.

---

time $t = 0$ causes the system to cool towards the ambient temperature as:

$$T = T_{amb} + (T_{jmax} - T_{amb})e^{-\frac{t}{R \cdot C}} \tag{3.7}$$

Figure 3.5 shows this trend for the same thermal capacitance and resistance values as above. The temperature asymptotically drops towards T_{amb}, but the fall time is more gradual with higher thermal capacitance.

Thermal state of the system. The significance of Equation 3.6 is that for any given system (*i.e.*, for a fixed thermal resistance and capacitance), the future temperature in response to input power can be determined from only its previous temperature. Thus, the thermal state of such a system at any instant of time is characterized solely by its temperature

3.3 Thermal Transient due to Latent Heat of Phase Change

A second form of thermal capacitance manifests in materials that change phase, such as when solids melt, or when amorphous substances change to crystalline. When such a material attains its phase change temperature (*e.g.*, melting point), any further heat is absorbed by the phase-change process and results in no further increase in temperature until all the material has changed phase. The amount of heat required for 1 gram of material to change phase is called its *specific latent heat*. The

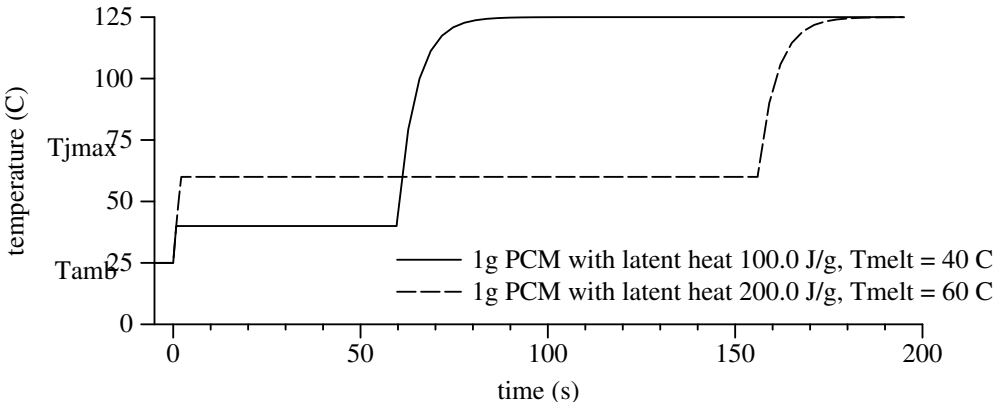


Figure 3.6: Thermal response with two different phase-change materials, for input power of 2 W and thermal resistance to ambient of 50 C/W in both cases. ❶, shows an example with 1 g of material with latent heat 100 J/g and melting point 40 $°C$. The temperature remains at 40 $°C$ for 58 s until the entire material melts, after which temperature rises again. In contrast, ❷ shows the thermal response of 1 g of PCM with latent heat 200 J/g, and melting point 60J$°C$, which requires 153 s to melt completely.

---

total latent heat for a given mass of material—the product of specific latent heat and mass, is thus the total energy absorbed for the entire material, and is symbolized as L.

3.3.1 Phase-change while heating

Figure 3.6 shows example thermal responses for two configurations: ❶ incorporates 1 g of phase change material (PCM) with specific latent heat of 100 J/g and melting point of 40 $°C$, and ❷ incorporates 1 g of material with twice the latent heat (200 J/g) which melts at 60 $°C$. (The example values are representative of the latent heat of fusion for paraffin). The thermal resistance and specific heat values are 50 $°C/W$ and 0.1 J/K as in previous examples.

Initially, the system is at its idle equilibrium state at ambient temperature. In response to an input power of 2 W being applied at time $t = 0$, the temperature rises with the R-C time constant as seen before. However, before heating up to the final equilibrium temperature of 125 $°C$, the material starts to melt when the temperature reaches the melting point. For ❶, the phase-change (melting) process begins at 40 $°C$. With a latent heat of 100 J/g, it takes 100 J of heat for the entire 1 g of material to melt. Therefore, for input power of 2 W, and heat dissipation rate of 0.3 W ($40\ °C - 25\ °C)/50\ °C/W$), the entire melting phase lasts for approximately 58 s (100 J/(2 W - 0.3 W)). For a different PCM ❷ with melting point at 60 $°C$, and latent heat of 200 J/g, the same

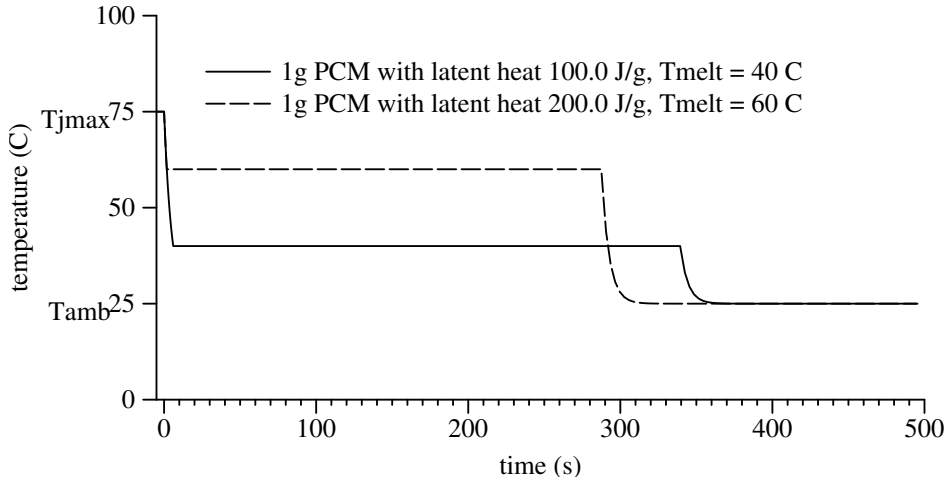


Figure 3.7: Cooldown time for two different PCMs starting from an initial temperature of 75°C, after which power is reduced to 0 W at time $t = 0$ for two different. Thermal resistance to ambient of 50 C/W in both cases. ❶ shows an example with 1 g of material with latent heat 100 J/g, which begins to freeze when temperature drops to the phase change (freezing) temperature of 40 °C. The temperature remains constant till the entire phase-change is reversed in 330 s. ❷ shows the same effect for 1 g of a PCM with latent heat of 200 J/g and freezing point at 60 °C. The freezing process lasts for 285 s. In both cases, after the phase-reversal is complete, the temperature asymptotically falls towards the ambient temperature.

---

amount of such material (1 g) melts in 153 s. Although the heat storage capacity is doubled from 100 J to 200 J, the duration of phase change is over 2.6× (58 s versus 153 s) because at a higher melting point the heat dissipation rate also increases due to the larger temperature gradient between the material and ambient air.

3.3.2 Phase-change While Cooling

Figure 3.7 shows the phase-reversal process for the same example PCM configurations as above when turning off heat (input power = 0) from an initial state where all the material has melted, and the system is at an equilibrium temperature of 75°C. The temperature drops according to the R-C rate until it reaches the melting point (now the freezing point) of the PCM, and remains at this value until all the phase-change has been reclaimed. For the example PCM with melting point of 40°C, heat is dissipated to the ambient environment at (40 - 25)/50 = 0.3 W. Therefore for the entire 100 J (total latent heat) to be dissipated, it takes 333 s. In the second case, the material starts freezing at

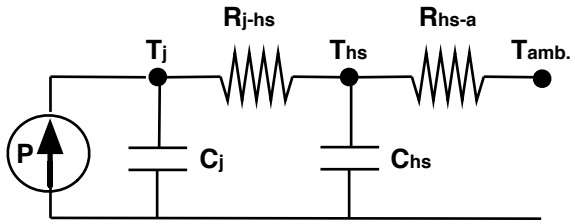


Figure 3.8: Representative thermal network with heat generated by processor operating with P, and dissipated to ambient environment via heatsink.

$60°C$ with a heat venting rate of 0.7 W, resulting in a freezing time of 285 s before the entire 200 J of its stored latent heat can be dissipated.

The phase-reversal time thus depends on the amount of latent heat stored in the material, and the rate of heat dissipation (thermal resistance and melting point).

3.3.3 Thermal State of a System with PCM

The above graphs illustrate that unlike a system with specific heat capacity alone, the future state of a system that undergoes phase-change cannot be determined solely by temperature. In fact, temperature is invariant during the phase-change process—that the temperature be equal to the melting point is a pre-condition to occurrence of phase-change. Rather, the thermal state of the system is now characterized by the residual latent heat, *i.e.*, how much of the material has melted.

If the stored (residual) latent heat at time t is $L(t)$, and the input power is P heat dissipation rate is P_{out}, then the future thermal state of the system is simply the difference in stored latent heat and the heat exchanged with the environment in the interval:

$$L(t + \delta t) = L(t) - (P - P_{out}) \cdot \delta t \tag{3.8}$$

If the system has been heating up in the interval ($P > P_{out}$) and all the material has melted ($L(t + \delta t) = 0$), then the phase-change process stops. Similarly, the phase-change stops when the system has been cooling ($P < P_{out}$) and all the material has frozen ($L(t + \delta t) = L_{total}$). Any subsequent heating or cooling causes a corresponding increase or decrease in temperature according to the R-C equation seen in Section 3.2.

3.4 Modeling a System using a Thermal R-C Network

This dissertation uses thermal models for simulation-based feasibility studies of sprinting, and to explain the thermal response observed on the sprinting testbed (Chapter 5. In either case the objective of the model is to estimate the thermal state of the system at any instant. As described in Section 3.2 and Section 3.3, the thermal state of a single element is either just its temperature (when there is no phase-change), or the amount of phase-change that has previously occurred (during the phase-change process).

To derive such a thermal model, first consider the heat flow in a practical system. Typically, heat flows from the processor to the ambient air via multiple components, such as in a desktop computers with a heatsink mounted on a processor socket, or in a mobile phone with a case enclosing the processor. Even within the chip, heat generated by the operation of individual functional units spreads laterally across the die, and vertically across several semiconductor and metal layers. Prior work has used finite element analysis to derive thermal models which capture a variety of details to highlight local hotspots in larger out-of-order processors [69]. However, manycore chips with less power-hungry cores have been shown to be less prone to local hotspots [107]. Hence, this dissertation follows previous techniques of modeling the processor as a uniform heat source, with the heat dissipation path via components such as the case/heatsink being represented by thermal resistances [110]. Further, each of these components is also associated with a specific heat (and latent heat, where appropriate). Figure 3.8 shows an example of such a multi-stage thermal R-C network; such a multiple-element system can be analyzed by extending the above principles of a single R-C network to each R-C stage.

For example, if the junction in Figure 3.8 is associated with a PCM which melts at T_{melt}, then its thermal state at time t is composed of its temperature $T_j(t)$ $^\circ C$, and its stored latent heat $L(t)$ joules. Its thermal response to an input power P watts for a time interval of δt seconds, *i.e.*, its new thermal state at time $t + \delta t$ characterized by $T_j(t + \delta t)$ and $L(t + \delta t)$ can be estimated as:

$$\begin{cases} T_j(t+\delta t) & = T_{melt}, \\ L(t+\delta t) & = L(t) - (P - P_{out})/R, & \textbf{w}\text{hen melting or freezing,} \\ \\ T_j(t+\delta t) & = T_j(t) + \frac{\delta t}{R_{j\cdot hs}\cdot C_j} \cdot (P \cdot R_{j\cdot hs} - & T_j(t) + T_{hs}(t)), \\ L(t+deltat) & = L(t), & \textbf{o}\text{therwise} \end{cases}$$

The thermal state of other components such as the heatsink (T_{hs} and L_{hs}) can be similarly derived.

3.5 Thermal-aware Computing

Although dark silicon highlights the need for power and temperature mitigation with renewed urgency, a large body of prior work has examined thermal management because: (i) thermal design power can exceed worst-case operating power from pathological "power-viruses," (ii) large out-of-order processors can be subject to thermal hotspots because of power-hungry structures, and (iii) process variation can lead to chips with varying temperature profiles. Kong *et al.* [91] provide an extensive survey of the work in this domain. The discussion below summarizes related work in thermal-aware architectures/software and mentions features of such work that have been adapted in this dissertation.

3.5.1 Dynamic Thermal Management

Prior mechanisms to prevent thermal emergencies can broadly be divided along two dimensions: (i) based on function, *i.e.*, monitoring versus control and (ii) based on invocation (software/hardware). Thermal sensing is usually performed in hardware and exposed to software, whereas control policies may be implemented in hardware, software, or across both hardware and software.

Thermal Monitoring. Hardware designs introduced thermal monitors recognizing the large gap between common-case and worst-case power consumption [62]. Modern techniques either use on-die thermal sensors [62, 175] to measure temperature or estimate temperature based on chip activity extrapolated using power/thermal models [97, 115]. On-die thermal sensors directly measure chip temperature either using thermistors or using thermal diodes in voltage or current mode, and are typically exposed to software using registers. An alternative scheme couples operating power with a thermal model of the chip to estimate temperature. Power is determined either directly through

on-chip power-meters or indirectly by correlation with instruction execution and performance counters [75, 97].

This dissertation uses instruction execution for temperature estimation for the simulation-based experiments in Chapter 4, directly reads temperature from the testbed machine [9] for several of the sprinting experiments in Chapter 5, and monitors power/energy for dynamic sprint-pacing in Section 6.5. In the PCM-based experiments, the thermal state of the system is characterized not only by temperature but also by the latent heat stored in the PCM at any instant; periodic energy monitoring can thus estimate the thermal state of a PCM-augmented system.

Hardware-based Thermal Management Policies. Initially, the focus of hardware-based thermal management was on preventing emergencies by triggering frequency throttling, reducing clock duty-cycle, and ultimately shutting off the system [18, 62]. Other work expanded the use of thermal feedback to mitigate performance and energy loss. Brooks and Martonosi evaluate schemes to control instruction-level parallelism by throttling pipeline stages in response to temperature [29]. Skadron *et al.* [154] propose micro-architectural techniques including DVFS and migrating computation based on fine-grained thermal monitoring. In multicore SMT processors, Powell *et al.* [132] propose heat-and-run as a policy where threads are aggregated to a single SMT core and migrated when the temperature reaches a threshold. Several previous works have also proposed selecting the operating power of a processor by setting voltage and frequency based on temperature and examined energy-performance trade-offs in this context [29, 33, 43, 123].

Software-based Thermal Management Policies. The operating system scheduler can influence die temperature through several means because chip activity, and hence temperature, is essentially driven by task execution. Kumar *et al.* [94] use estimated power consumption per task as a scheduling priority input. For multicore environments, Choi *et al.* [36] propose per-core task assignment based on temperature. Other core-hopping techniques function similar to heat-and-run, where the OS migrates threads to avoid cores from overheating. Recent work explored implementing a thermal-aware scheduler for Linux and showed that the average time spent at peak temperature could be reduced [178]. Even more recently, Intel announced a thermal daemon for Linux to prevent BIOS-level thermal throttling by proactively selecting fan-speeds based on measured activity [2].

The sprinting runtime used in this dissertation (Chapter 5) uses temperature-based DVFS settings similar to some of the above proposals. Similar to OS-based throttling, sprint-termination is implemented by reducing frequency and by deactivating cores.

3.5.2 Thermal-aware Design and Layout

Researchers have also proposed thermal-aware floorplanning to mitigate the appearance of hotspots. The primary insight in many of these approaches is to locate and place sources of heat close to "cold spots" to facilitate heat spreading and more uniform die temperatures. The HotSpot tool by Skadron *et al.* [154] is widely used by academics to estimate temperature based on floorplan. Powell and Vijaykumar [133] evaluate enlarging hot functional units to increase heat spreading without compromising delay along critical paths. Floorplan studies have led to key inferences about multi- and manycore architectures, as well as emerging 3D architectures. Moncheiro *et al.* [120] explore multicore layouts and find that laying out the cores in the center of the chip, with caches around them leads to lower average temperature because of improved heat spreading. Puttaswamy and Loh [134] propose thermal herding in stacked-3D multicores so that execution is driven towards the die nearest to the heat-spreader.

This dissertation evaluates sprinting using multiple cores, and treats the entire die as a uniform heat source based on these works that suggest that hotspots can be mitigated using circuit layout techniques. Huang *et al.* [71] observe that manycore architectures with small cores tend not to suffer from hotspots because the heat generated by the cores is effectively absorbed in the space between cores.

Based on the thermal principles presented in this chapter, the next chapter investigates the feasibility of sprinting by exceeding sustainable power temporarily.

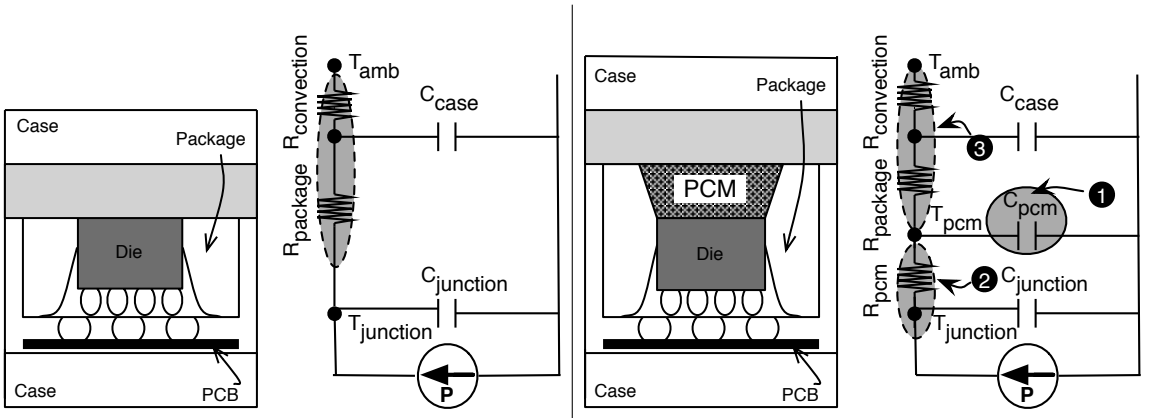
Chapter 4

Feasibility Study of Computational Sprinting

Computational sprinting is motivated by three observations from Chapter 1 and Chapter 2: (i) sustainable performance will be limited by thermal conductivity, especially in mobile devices, (ii) interactive applications demand responsiveness—intense, brief bursts of computation punctuated by longer durations of idleness, and (iii) thermal capacitance can buffer heat to allow temporarily exceeding sustainable power. Based on the estimates of dark silicon in Chapter 2, which indicate a 10× gap between sustainable and peak power, this chapter investigates sprinting to enable 10× improvements in responsiveness for interactive applications.

As a concrete objective, this chapter considers *parallel* computational sprinting, in which a system which can sustain the operation of a single 1 W core is enabled to sprint for up to one second by activating 15 otherwise "dark" cores. To recap sprinting operation from Section 1.2, computational sprinting begins with activating all 16 cores in response to an input event. The heat in excess of the sustainable dissipation rate of the system (1 W) is buffered by the thermal capacitance in the system. After exhausting this thermal buffer, when the temperature nears the permissible threshold, the system stops sprinting and completes any remaining computation at sustainable power, *i.e.*, by powering off the 15 additional cores and executing all instructions on a single core.

Section 4.1 first illustrates a strawman-proposal to buffer the 15 W of excess heat for one second to sustain one such sprint. This section evaluates the approach presented in Section 1.2—augmenting thermal capacitance with latent heat of phase-change—by extending the familiar ther-



(a) Phone cut-away (b) Thermal network (c) PCM placed near die (d) Thermal network with PCM

Figure 4.1: The thermal components of a mobile system (a) and its thermal-equivalent circuit model (b). In (c) and (d), the system is augmented with a block of phase change material (PCM). The amount of computation possible during a sprint is primarily the system cools after a sprint.

mal analysis techniques from Chapter 3 to physical constants representative of a mobile phone [110]. To motivate the potential benefits of sprinting, Section 4.2 then evaluates the responsiveness benefits that such parallel sprints can enable for sample applications executed on a simulated manycore sprinting system. Section 4.3 extends the evaluation to a rudimentary model of repeated sprints separated by think times.

Following the quantitative introduction of the approach and potential benefits of sprinting, this chapter examines the feasibility of engineering a mobile system capable of sustaining such a 16-core sprint for 1 second. The feasibility analysis broadly considers the immediate thermal (Section 4.4), electrical (Section 4.5) and hardware/software challenges (Section 4.6) imposed by sprinting on existing systems and proposes approaches to address these challenges. Although not the focus of this dissertation, this feasibility study briefly discusses the implications of sprinting on reliability (Section 4.7) and cost (Section 4.8).

4.1 A Thermally-augmented System for Sprinting

To understand the basic thermal approach to sprinting, consider the heat flow in an example mobile phone. Figure 4.1a shows the physical arrangement of a package containing the processor die inside a mobile phone case. The thermal R-C network in Figure 4.1b represents the heat-path between the

Parameter	Description	Value
$R_{package}$	Chip to case thermal resistance	12.1 K/W
$C_{junction}$	Thermal capacitance of die	0.011 J/K
$R_{convection}$	Case-to-air thermal resistance (convection)	28 K/W
C_{case}	Thermal capacitance of case	8.3 J/K
R_{PCM}	Thermal resistance between die and PCM	0.001 K/W
C_{PCM}	Thermal capacitance of PCM (latent heat)	100 J/g
T_{amb}	Ambient temperature	$25^{\circ}C$
T_{jmax}	Maximum junction temperature	$75^{\circ}C$

Table 4.1: Thermal model parameters

processor and the ambient air. The parameter values (Table 4.1) are derived from a physically validated model of a mobile phone from 2008 by Luo *et al.* [110] (subsequent estimates by Shao *et al.* [149] show similar values). This study used temperature probes to show that heat flows from the processor to ambient environment mostly along two paths: an upper path through the circuit board and top surface of the phone and a lower path through the battery and bottom surface, and that these two parallel paths should optimally have the same thermal conductivity. Lumping these parallel paths from the chip to the case yields the thermal network model in Figure 4.1b.

4.1.1 Thermal Resistance, Thermal Design Power, and Thermal Capacitance

The processor die operates at power P and is considered as a uniform heat source with temperature $T_{junction}$ measured at the junction. The thermal capacitance at the junction ($C_{junction}$) is from the processor's thermal mass (*i.e.*, from its specific heat). The other significant source of thermal capacitance is the case (C_{case}). Heat spreads from the junction to the case through the package thermal resistance ($R_{package}$), and the case itself dissipates heat to the ambient environment (which is at temperature T_{amb}) due to passive convection, which is represented by its equivalent thermal resistance ($R_{convection}$).

Thermal design power. For a phone operating under typical conditions, a representative value for the total thermal resistance from the die to the ambient environment is 40 K/W. Assuming an initial chip temperature of 25° C (room temperature), and a maximum temperature of 75° C, the maximum sustainable power is therefore 1.25 W ((75 - 25)/40). If the initial die temperature is higher than the room temperature, the thermal design power of this system would decrease; for example, the TDP would be 1 W for ambient temperature of 35° C (approximately body temper-

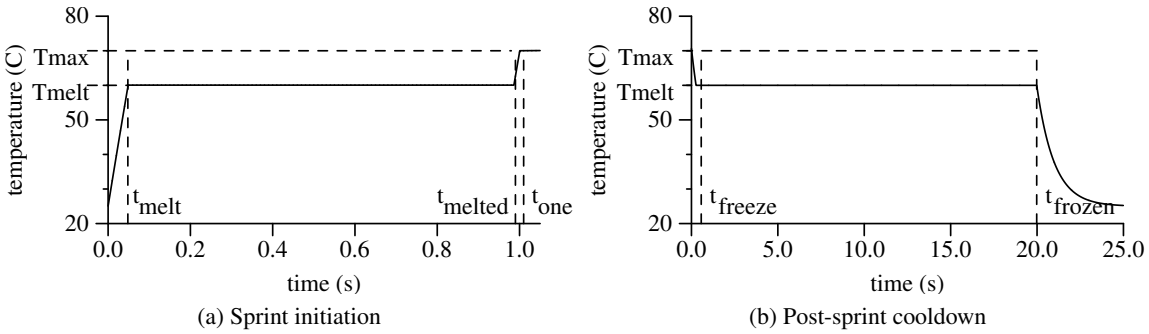
ature/temperature of a human hand holding a phone). In the context of this evaluation, where the processor consists of 1 W cores, the mobile system can sustain execution of only one such core.

Thermal capacitance. Consider a mobile processor with a die area of 64mm^2 (comparable to today's mobile processors). Given the specific heat of silicon (0.7 J/gK), the thermal capacitance from such a die is only 0.01 J/K. Hence, during a 50°C temperature increase starting from room temperature of 25°C to a threshold value of 75°C, the die absorbs only 0.5 J of heat. The heat buffering from the die itself is hence sufficient to allow less than 35ms of sprinting with 15 additional 1 W cores. Although a mobile phone case weighs over 100 g (with specific heat of 8.3 J/K), and can thus absorb sufficient heat for a single sprint, the poor thermal conductivity between the die and case prevents effective use of such thermal headroom. (Heat flows between the die and case at the rate of only 1 W for a 12°C temperature gradient—see Table 4.1)

4.1.2 Sprinting on a System Augmented with Phase Change Material

As seen in Section 1.2, one approach to buffer the excess heat is to augment the system with the latent heat of phase-change. Phase-change materials are attractive as sources of thermal capacitance due to the large amount of heat absorbed during the phase-change process. Figure 4.1c shows a potential design for the mobile system from Figure 4.1a augmented with a phase change material placed inside the package. For the moment, assume the material's melting point to be 60°C, with near-ideal heat spreading within the PCM manifesting as a small thermal resistance in the equivalent thermal network in Figure 4.1d. Section 4.4 discusses the practical considerations of phase-change materials (including heat spreading) in more detail. Considering a PCM with a latent heat of 100 J/g and density of 1 g/cm^3, about 150 milligrams of PCM (2.3mm thick block of PCM in contact with a 64mm^2 die) can absorb approximately 16 J of heat, which is enough for 16 cores to operate at 1W each for 1s at a constant temperature.

In addition to the thermal resistance between the die and PCM (R_{PCM}) and the thermal capacitance of the PCM (C_{PCM}) which determine sprint intensity and duration, the interval between successive sprints is limited by the ability of the PCM to re-solidify—the rate of heat transfer between the PCM and the ambient environment ($R_{package}$ + $R_{convection}$). These thermal transients are evaluated next.



(a) Sprint initiation

(b) Post-sprint cooldown

Figure 4.2: Transient behavior of initiation and termination of sprinting

4.1.3 Temperature Transients of Sprinting with Phase-change Change Materials

Figure 4.2a illustrates the transient thermal response of a 16W sprint on a 1W TDP system under the thermal model and PCM assumptions described in the previous section. Junction temperature initially rises rapidly, then plateaus for 0.95s during the phase change, subsequently rising to the maximum safe temperature (set at 70°C for these simulations). Factoring in all sources of thermal capacitance, the sprint can be sustained for a little over 1s.

Heat flux considerations. Although the estimates of the peak heat flux (25 W/cm^2) and rate of temperature increase (35°C in approximately 50ms) from Figure 4.2a is high for a mobile device, they are still below the typical range for high-end processors [114]. Two implications follow. First, the mechanical stresses incurred due to heating during a sprint are not extraordinary and should not fundamentally curtail the lifetime of a sprint-enabled chip. Second, the required thermal conductance between the junction and PCM falls within the range of conventional thermal interface materials (TIMs), and is therefore not expected to limit sprint intensity. Although such a peak heat flux is perhaps not a fundamental barrier to feasibility, it could necessitate a more expensive package. Integrating the PCM into the package and placing it close to the die would limit the greater heat flux to only the TIM (since heat can be released from the PCM over a longer time period between sprints), thus ameliorating thermal demands on the package as a whole.

Cooldown interval. Between sprints, the PCM returns to its original phase as the system cools back down to the initial temperature, dissipating heat at the sustainable rate (TDP). Approximate cooldown duration can hence be calculated by multiplying the duration of the sprint by the ratio

Processor	in-order x86-64, 2W-peak, 1W-average, 1GHz, 32KB 8-way L1, 64B blocks
Last-level cache	shared 4MB, 16-way, 64B blocks, 20 cycle hit latency, directory cache coherence protocol
Memory	Dual-channel 4GB/s LP-DDR2, 60ns round-trip latency
Normalized Power	Int ALU: 0.22, FP ALU: 0.66, memory access: 1.0, stall: 0.1
Thermal model	junction capacitance: .011 J/K, junction-to-case resistance: 12.1 K/W, case capacitance: 8.3 J/K, convection resistance: 25 K/W, PCM: latent heat 100 J/g, melting point: 333 K, mass: (a) 0.003 g, (b) 0.3 g, sampling frequency: 1000 cycles

Table 4.2: Simulator parameters

of sprint power and nominal TDP. Figure 4.2b shows the cooldown behavior based on the above thermal model. The exponential nature of heat transfer results in the junction temperature reaching close to nominal after approximately 24s. The cooldown period is governed primarily by the amount of thermal capacitance, the thermal resistances between the PCM and the ambient environment (see Figure 4.1), and to a lesser degree by the melting point of the PCM. The higher the melting point of the PCM, the larger the temperature gradient between the PCM and ambient, which accelerates cooling (see Chapter 3.

4.2 Architectural Evaluation

To evaluate the potential energy benefits of a sprinting system, this section simulates a suite of kernels representative of the computation in emerging mobile applications. To estimate runtime (performance) and energy consumption, these kernels are executed on a simulator which approximates the architectural (performance, energy) and thermal characteristics of a future many-core mobile device. The evaluation below analyzes the sensitivity of sprinting to compute demand (by varying a range of input sizes), thermal design points, and number of cores used to sprint.

4.2.1 Simulation Methodology

To show the feasibility and utility of parallel sprinting, this evaluation uses a many-core instruction-level simulator to model sprint initiation and system behavior when the sprint interval is exhausted. The simulator accounts for performance by counting execution cycles and associates a fixed dynamic energy to each type of instruction executed. The simulator determines thermal state by periodically invoking a thermal network simulator with operating power (operating power is derived

from the sampled dynamic energy every interval). When the temperature nears T_{max}, the simulator *truncates* sprinting by executing all threads on a single core and deactivating any other active cores.

Covering a full sprint requires many-core simulations on timescales of up to one second—billions of instructions (1 billion cycles for 16 1-IPC[1] cores at 1 Ghz is 16 billion instructions). The parallel nature of the software and the need for periodic thermal monitoring (to which the runtime must immediately react) preclude the use of sampling or other simulation acceleration techniques in a straightforward manner. To allow tractable simulation times of these timescales, the simulator models in-order x86 cores with a CPI[2] of one plus cache miss penalties. The cores have private 32KB 8-way set-associative caches, a shared 4MB 16-way last-level cache with 20 cycle hit latency, and a dual-channel memory interface with 4GB/sec channels and an un-contended 60ns round-trip latency. The memory system models a standard invalidation-based cache coherence protocol with the directory co-located with the last-level cache. When sprinting begins, the L1 caches are initially empty and the cores are enabled only after the power supply is stable.

This performance model is augmented with a dynamic energy model that associates energy with the type of instruction being executed. The energy estimates are derived from McPAT [105], configured at a 1GHz, 1W core, 22nm LOP (low operating power) technology node. During execution, energy consumed by each core is sampled every 1000 cycles to drive the thermal RC network model of the PCM-augmented heatsink described earlier in Section 4.4. The chip is assumed to be a uniform heat source and does not account for temperature gradients across the die; recent work has shown tiled many-core architectures to be less susceptible to hot-spots [72]. To mitigate energy losses due to load imbalance and busy-waiting [102], the software runtime inserts `PAUSE` instructions on barriers, spinning on locks, and repeated failed task-stealing attempts. Upon encountering a `PAUSE` instruction issued by a thread in sprint mode, the hardware puts that core to sleep for 1000 cycles. The dynamic power dissipation of a sleeping core is modeled as 10% that of an active core. Table 4.2 lists the simulator parameters.

4.2.2 Workloads

The workloads used in this evaluation are intended to be representative of the computation performed by vision-oriented interactive workloads such as those used in computational photography.

[1] instructions per cycle
[2] cycles per instruction

Kernel	**D**escription
`sobel`	Edge detection filter; parallelized with OpenMP
`feature`	Feature extraction (SURF) from [38]
`kmeans`	Partition based clustering; parallelized with OpenMP
`disparity`	Stereo image disparity detection; adapted from [168]
`texture`	Image composition; adapted from [168]
`segment`	Image feature classification; adapted from [168]

Table 4.3: Parallel kernels used in the evaluation

- **Feature extraction.** `feature` from MEVBench [38] is a feature extraction application that is representative of the processing performed in camera-based search [58]. The distributed MEVBench implementation of SURF (Speeded-Up Robust Features) uses pthreads for parallelism and is executed almost unmodified in this evaluation (barring static linking and simulator callbacks for timing). The input region is divided vertically and horizontally into regions which are handed to worker threads. Each worker thread first localizes feature points, then builds descriptors, next redistributes the extracted information conditionally, and finally writes back the extracted data. After each of the above steps, threads synchronize at a barrier. Length of computation varies with image size.

- **Disparity comparison.** `disparity` from SD-VBS [168], computes a disparity map that can be used to derive the relative depth of objects in a scene given two stereo images of the scene. The input to the workload is a pair of images (left and right) taken from slightly different positions, and a "shift" window. The sequential implementation shifts the right image by each shift value, correlates it with the left image (pixel-by-pixel), finding the minimum disparity between the images. The parallel implementation computes the disparity map for each shift window across the 2D-images in parallel, synchronizing in the last step to combine the minimum value. Length of computation varies with image and window size.

- **Edge detection.** `sobel` is an edge detection filter which convolves a 3×3 (constant) matrix across an input image. The computation is hence both regular and independent of pixel values. The parallel implementation tiles the image and applies the convolution step independently to each tile using OpenMP for the outer `for` loop. Computation length varies with image dimensions.

- **Clustering.** `kmeans` assigns an array of n points, each with d dimensions, to k clusters such that the sum of distances between each point and the center of its cluster (mean) is minimum. This kernel is used in vision applications like image segmentation. The sequential algorithm first assigns the first k points to be cluster centers, then iteratively recomputes the nearest cluster for each point, recomputing the new center of a cluster if its membership has changed. Based on previous implementations used as architectural benchmarks [86, 116], the "for-each-object" loop is parallelized, with thread-local clusters per-thread which are aggregated (array reduction) by the main thread. Computation length varies with number of clusters, points and dimensions.

- **Texture synthesis.** `texture` takes as input a small image and creates a larger image that is similar in texture to the small image. Such algorithms are used to fill holes or blank spots in image regions that exhibit some uniformity. The sequential implementation from SD-VBS [168] fills the output image dimensions from left to right and top to bottom scanning each pixel for candidate matches from the input texture and neighboring regions (already filled) in the output image. The parallel implementation tiles the image into rectangular blocks with each block filled independently. However, a caveat is that the filling proceeds with a neighboring ''L-shaped" causality — blocks can only be processed if their preceding tiles have been filled.

- **Segmentation.** `segment` from SD-VBS [168] partitions an image into segments that share similar visual characteristics. The sequential implementation blurs the input image using an edge detection kernel (similar to `sobel`), next assigns edge weights between neighboring pixels based on the smoothed image to create a graph, then sorts the edges based on these weights and finally creates clusters by collapsing nodes that share an edge if the edge weight between nodes (pixels or collapsed clusters) is less than an acceptable threshold (similar to `kmeans`). Each of the pieces is task-parallelized. Compute length varies with image size.

Although these workloads are not complete applications, they represent the computation stages performed by interactive applications. Edge detection, clustering, and segmentation are preliminary steps in image processing and analysis, pattern recognition and several computer vision techniques. Clustering (`kmeans`) is used not only for signal processing, but also for applications such as su-

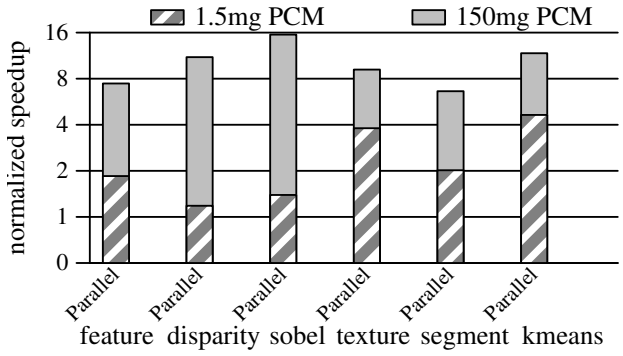


Figure 4.3: Parallel speedup on different workloads for 16 cores.

pervised and semi-supervised feature/dictionary learning. Segmentation and feature extraction aid interactive applications like face and character recognition. Chapter 6 additionally introduces a speech recognition workload (not included here due to vagaries of the simulator). With human-computer interaction becoming increasingly prevalent in mobile devices (especially phones and tablets) phones and tablets, these workloads together capture some of the computation expected of such applications, but also some tasks which are today offloaded to the "cloud" due to computational complexity (for example `feature` performs some of the computation representative of Google Goggles). However, because their most common uses entail human activity, performing such computation locally (on the mobile device) within human-acceptable response times could potentially improve user experience, and further enable new applications composed of such computation kernels.

4.2.3 Increased Responsiveness

To demonstrate the improvement in responsiveness (*i.e.*, speedup in processing time to complete a short task), the parallel workloads were simulated on 16 cores with the full-provisioned thermal configuration (150 milligrams of PCM). The total height of the bars in Figure 4.3 shows an average parallel speedup of 10.2× over a single-core non-sprinting baseline. The sprinting and non-sprinting configurations both have the same TDP, last-level cache, and memory bandwidth constraints, but the sprinting configuration is able to use the additional cores (assumed to have been dark silicon) to improve responsiveness.

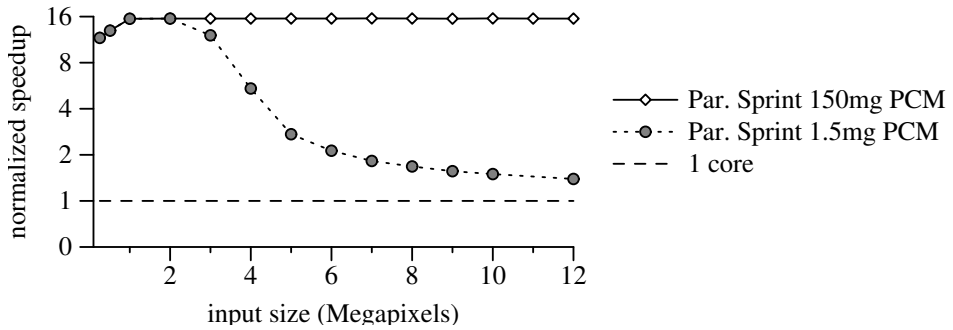


Figure 4.4: Parallel speedup for `sobel` with increasing computational demand for 16 cores compared with idealized DVFS with the same maximum sprint power.

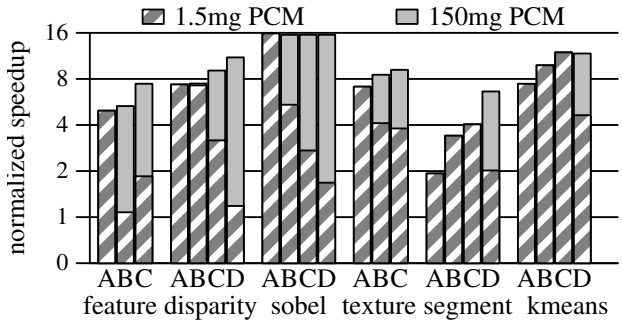


Figure 4.5: Speedup on 16 cores with varying input sizes

4.2.4 Thermal Capacitance Design Point

To measure the effect of limited sprint duration with tractable simulation times, the thermal capacitance of the system is reduced to 1% of the fully provisioned PCM. The bottom segment of the bars in Figure 4.3 represent the speedup for this design point. These simulations show smaller speedups, because under this more constrained thermal configuration all workloads exhaust the sprint duration and are forced to execute some of the computation in the post-sprint single-core mode.

Figure 4.4 shows the impact of the thermal design on speedup (y-axis) for the `sobel` kernel as the amount of computation per sprint is increased with image resolution (x-axis). Excepting the lowest resolution images, `sobel` scales linearly up to 16 cores. For the fully sized PCM, the system is able to sustain the sprint for the entire computation at all image resolutions. However, for the artificially limited design point (1.5mg of PCM), the graph shows that speedup drops off as the fixed-sized sprint can handle less of the total computation.

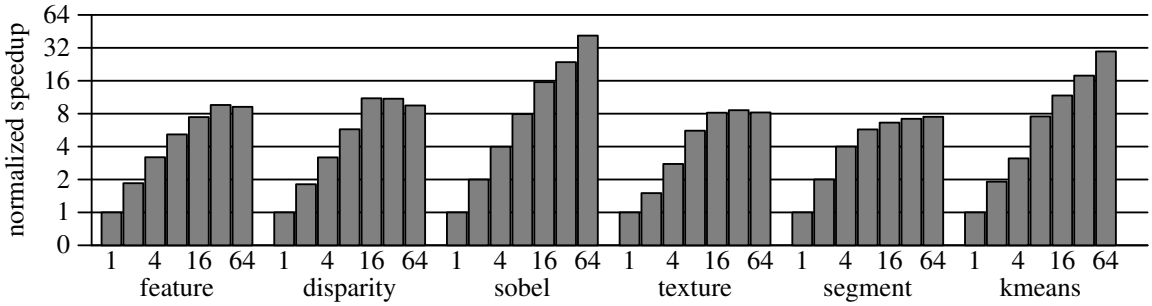


Figure 4.6: Parallel speedup with varying core counts at fixed voltage and frequency

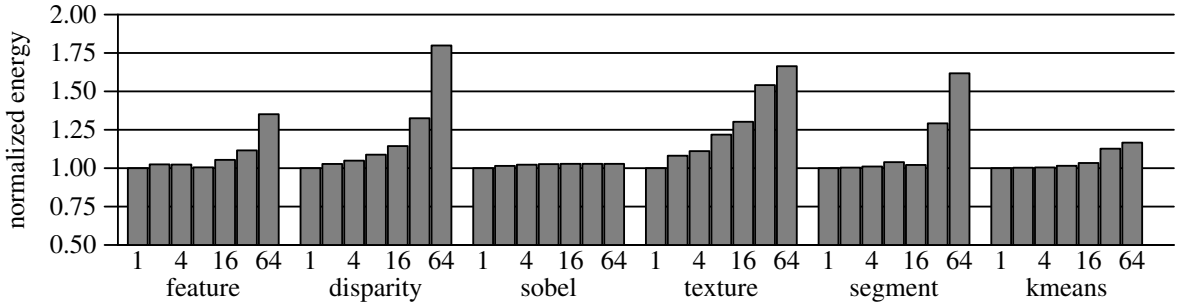


Figure 4.7: Dynamic energy with varying core counts at fixed voltage and frequency

Figure 4.5 shows speedups for all the workloads with varying problem sizes, reinforcing the trend of larger problems sizes exhibiting higher parallel speedup, but also requiring larger thermal capacitance to complete during the sprint window. As seen in the `feature` application, parallel sprinting achieves an 8× speedup with the largest input size (HD image, bar C)—which allows the user to process an image with 8× the amount of detail than would be possible in a traditional (non-sprinting) device.

4.2.5 Varying Intensity of Parallel Sprinting

The results reported thus far assume 16 cores. However, changes in scaling trends could result in fewer or more cores available for sprinting on a future chip. Figure 4.6 shows how changing the number of sprinting cores affects the responsiveness (speedup) of each workload (for largest input size) over a single core baseline. As expected, configurations with fewer cores exhibit smaller speedups but are able to extract a higher percentage of peak throughput. With more cores, scaling

diminishes, but `kmeans` and `sobel` continue to scale well up to 64 cores. The scaling of the other workloads are limited either by the available parallelism or by available memory bandwidth. A more detailed characterization of these overheads follows the dynamic energy analysis below.

4.2.6 Dynamic Energy Analysis

Figure 4.7 shows the total dynamic energy consumption for each workload with varying numbers of cores. When operating in a region of linear speedup, the dynamic energy of the parallel sprint is unsurprisingly similar to the dynamic energy of single-core execution—the same amount of work is performed by many cores in proportionally less time. Even when operating in modes beyond linear scaling (*e.g.*, 6.6× speedup on 16 cores for `segment`), the runtime system's use of sleep modes is effective in avoiding dynamic energy increases. On 16 cores, the energy overhead due to parallel sprinting is less than 10% on five out of six workloads and only 12% on average. However, beyond sixteen cores, non-linear scaling and parallel execution overheads result in energy overheads of up to 1.8× over sequential execution (`disparity` on 64 cores).

4.2.7 Instruction Overheads

To characterize the cause for energy overheads (and lack of scaling), Figure 4.8 first shows the extra instructions introduced in parallel execution. In `feature`, `disparity`, `texture` and `segment`, additional instructions are executed to set up each thread's region boundaries. The `feature` workload is also heavily barrier synchronized (four barriers) which increases dynamic instruction count—although the `pause` instructions introduced at barriers mitigate busy-waiting by idling cores for 1000 cycles, the overhead grows significant when threads are mostly idle (Figure 4.9). The instruction overhead is increasingly pronounced for the higher core counts in `disparity`, `texture` and `segment` where the number of threads exceed the available parallel computation (beyond 16 threads); the resulting overheads when threads execute wasteful instructions when looking for computation to execute can be avoided by aggressively throttling down idle threads.

4.2.8 Runtime Breakdown

To discern the overheads still further, Figure 4.9 divides fractions of the total execution time for each configuration and workload based on processor activity (aggregated from all cores). The "compute"

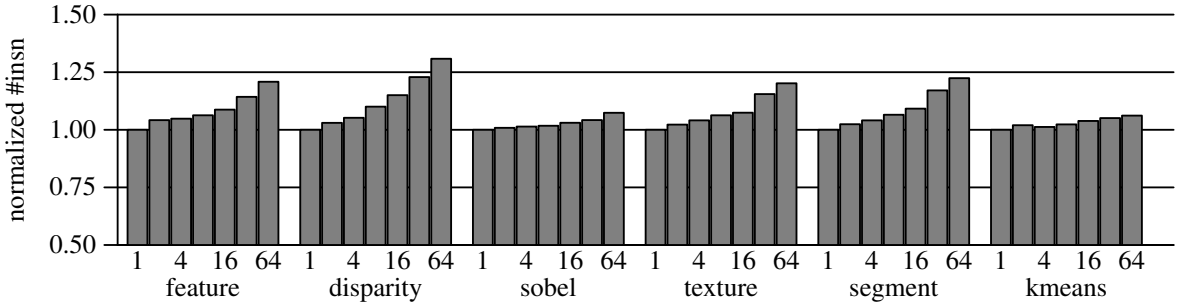


Figure 4.8: Instruction overhead with increasing core count

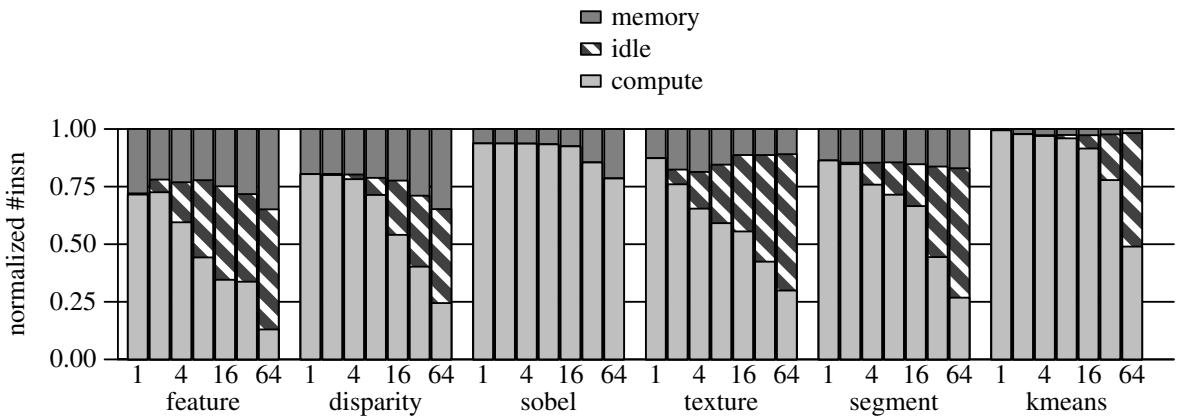


Figure 4.9: Percentage time breakdown

component represents actual instruction execution, "memory" denotes the time spent waiting for cache misses and the middle segment denotes the fraction of time when a processor is "idle" (when a core is put to sleep after executing a `PAUSE` instruction). For `feature`, `disparity` and `sobel`, increasing the number of cores beyond 32 causes the memory fraction to grow because of bandwidth pressure. Doubling the memory channel bandwidth caused the speedup of all these workloads to increase (`sobel` to 53×, `disparity` to 11.5× and `feature` to 11.9×). However, the idle time is the clearly more significant component. In all of the workloads barring `sobel`, with 64 cores, greater than 50% of the time is spent in idle mode due to lack of parallelism. The additional energy overheads arise from the idle power (10% of dynamic power) associated with these periods in the energy model.

Evaluation Summary

The above evaluations show example applications where significantly exceeding thermal design power for brief stints of computation can enable order-of-magnitude performance improvements with little adverse impact on dynamic energy (10× average speedup when sprinting at 16× TDP for up to 1 second). Subsequent sections (Section 4.4, Section 4.5 and Section 4.6) investigate the feasibility of engineering a system capable of such sprints. However, the next section addresses *multiple* sprints, where a user interacts with a device repeatedly, albeit intermittently.

4.3 Multiple Sprints

Thus far, this evaluation only considered sprinting performance in the context of a single computational task. However, as seen in Section 4.1, after sprinting, a system must rest to recover its capacity to sprint again. The utilization of a sprinting device hence influences device performance. This section provides an initial, rudimentary exploration of the sensitivity of sprinting performance to: (i) device utilization (computation task size and task arrival rate) and (ii) thermal design parameters (available thermal capacitance).

In the absence of representative data for users adapting to sprinting devices in the future, this section makes several simplifying assumptions to model sprinting in response to varying computational load. Section 4.3.1 overviews the usage model of multiple sprints analyzed in this section, Section 4.3.2 identifies the sensitivity of responsiveness to computational demand. Section 4.3.3 then proposes a *thermal hierarchy* to capture the periodicity in bursts of computation.

4.3.1 Usage Model

Consider the following typical usage scenario of a mobile device such as a phone. A user intermittently operates the device, initiating computation tasks one at a time. Because these tasks are intermittent, they are separated by an interval modeled as a *think time* when the device is idle. The net computation demand on the device is hence characterized by the length of each computation task (*i.e.*, the per-task computation required) and the arrival interval (think time). Under conventional TDP-constrained operation, the peak-performance of the device is agnostic to the arrival interval because the execution time (at TDP) determines the response time of each task. However, when

sprinting, the system can speed up task execution depending on available thermal headroom. The response time of any given task hence varies with the thermal state of the system when it begins executing the task. Such a system resembles a closed-queueing system [16]; however because of the simplified request model (single customer, single service station with single-entry queue), and the dependence of service time on the thermal state of the system, the analysis below follows from straightforward simulations rather than direct application of queuing theory. As a further simplification, the performance of the system is assumed to be energy-proportional, *i.e.*, sprinting by a factor of N in power contributes to a speedup of N. This section therefore neglects any energy benefits from amortizing background power and energy penalties incurred to boost power, deferring such discussion to Section 6.3.

The above model allows for an initial exploration of the sensitivity of sprinting performance to system design and usage. The evaluation below varies the input parameters (computation length, arrival interval, sprint power/speedup and available thermal capacitance) independently. Comparing task-execution times of a sprinting system with a TDP-constrained system in each case then indicates the relative responsiveness benefits of sprinting.

4.3.2 How is Sprinting Performance Sensitive to Device Utilization?

As a concrete evaluation point, this section considers a system with single thermal component where the fixed thermal parameters resemble Table 4.1 (a 1 W TDP system comprising a die with specific heat capacity of 0.01 J/K, initial and final temperatures of 50°C and 75°C TDP, and thermal resistance to ambient of 25°C/W). Additionally, to examine the impact of thermal capacitance, consider a (variable) mass of PCM associated with the die (with ideal heat thermal conductivity between the die and the PCM). To model bursty computational demand, the next experiment simulates tasks with sizes and think times chosen from exponential probability distributions around a parameterized mean value.

Sensitivity to computational demand. First, consider the *computational demand* defined as the percentage of time that a non-sprinting, TDP-constrained baseline system spends computing. In the context of this evaluation, computational demand is hence $(\frac{\text{task size}}{\text{task size + think time}})$. In the limit, a computational demand of 100% represents sustained computation when the baseline is always busy, and a computational demand of 0% implies that the baseline system is always idle.

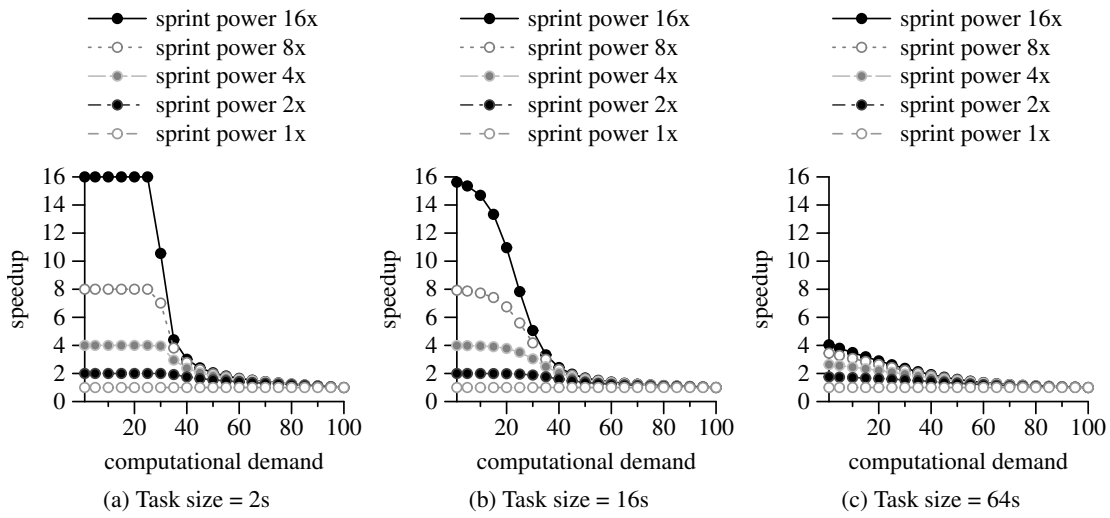


Figure 4.10: Sensitivity of responsiveness to varying computational demand for three different mean task sizes. Thermal capacitance is held constant.

Figure 4.10 shows the relative speedup under varying computational demand (*i.e.*, think time decreases from infinity to zero along the x-axis) when sprinting with fixed thermal capacitance (1 g of PCM). In Figure 4.10a, the mean task size is fixed to be smaller than the computation capacity of a single complete sprint (*i.e.*, starting from a state where the PCM is completely solid). Each line in the graph represents sprinting at a different intensity, with the 1× sprint power corresponding to a TDP-constrained baseline. For low computational demand (below 30%), the system can recover sufficiently between sprints to complete all tasks in sprint mode (evidenced by a speedup of N× with N cores). However, with increasing computational demand (*i.e.*, as think times decrease) the sustainable cooling rate of the system once again constrains performance. The different sprint intensities exhibit the same trend; because of the underlying energy-proportional computation, the total amount of computation performed for a given thermal capacity is invariant of power.

Figure 4.10b repeats the data from Figure 4.10a, but further increases computational demand *per-task* (mean task length of 16 s in Figure 4.10b as against 2 s in Figure 4.10a). The overall trend of speedup decreasing with growing computational demand still prevails. However, because each task now utilizes more of the thermal buffer, the fall in speedup is steeper than for the smaller task size. Increasing the mean task size further (64 s in Figure 4.10c) causes an overwhelming majority

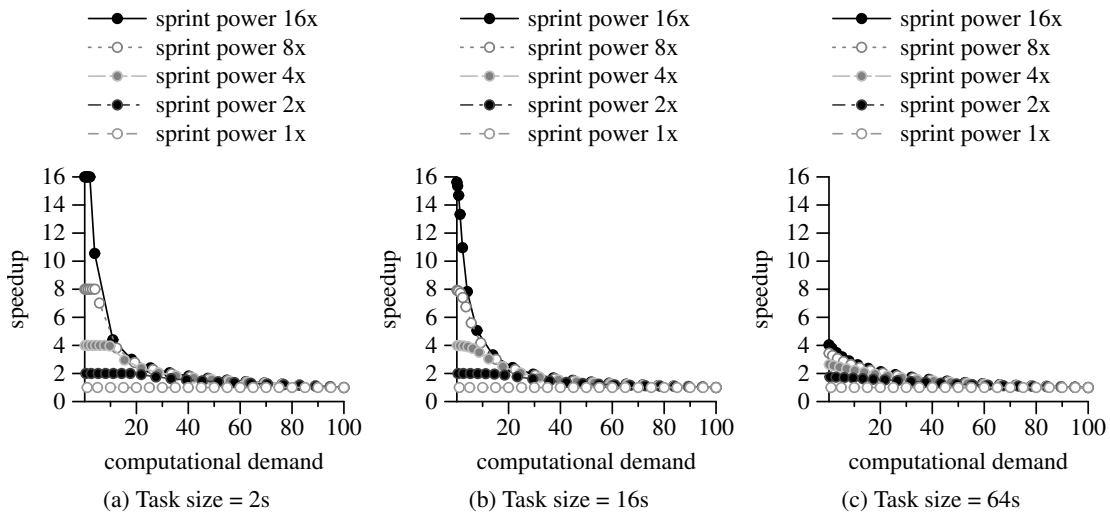


Figure 4.11: Speedup relative to a non-sprinting baseline with varying system utilization for different mean task sizes. Thermal capacitance is held constant.

---

of sprints to be truncated; this case is similar to the single sprints evaluated in Section 4.2 where speedup is limited by thermal capacitance.

Figure 4.11 renormalizes the above results to the *utilization* of each sprinting system. In this context, utilization is defined by the fraction of total time that the sprinting system executes computation (sprinting or truncated). As seen in Figure 4.11a, higher power sprints cause speedup to drop at proportionally lower utilization. Because the total energy available to sprint is constant across the different power sprinting configurations, higher power sprints entail a lower busy time to think time ratio (*i.e.*, a lower *duty cycle of operation*).

Effect of thermal capacitance. Figure 4.12 further clarifies the role of thermal capacitance on the performance of sprinting in response with varying utilization. For this set of experiments, the task size is held constant (64 s of baseline computation) and each graph represents sprinting with varying capacitance at a different intensity (4×, 8× and 16×respectively in Figure 4.12a, Figure 4.12b and Figure 4.12c). At one extremity, infinite capacitance allows the system to sprint on a sustained basis. At the other extremity, the extremely small thermal capacitance of the die allows for little sprinting. In between, as the thermal capacitance is increased by finite quantities, under low utilization the system is able to execute an increasing fraction of each task in sprint mode. However, speedup is still limited by utilization; because the heat stored in the system only vents at the rate

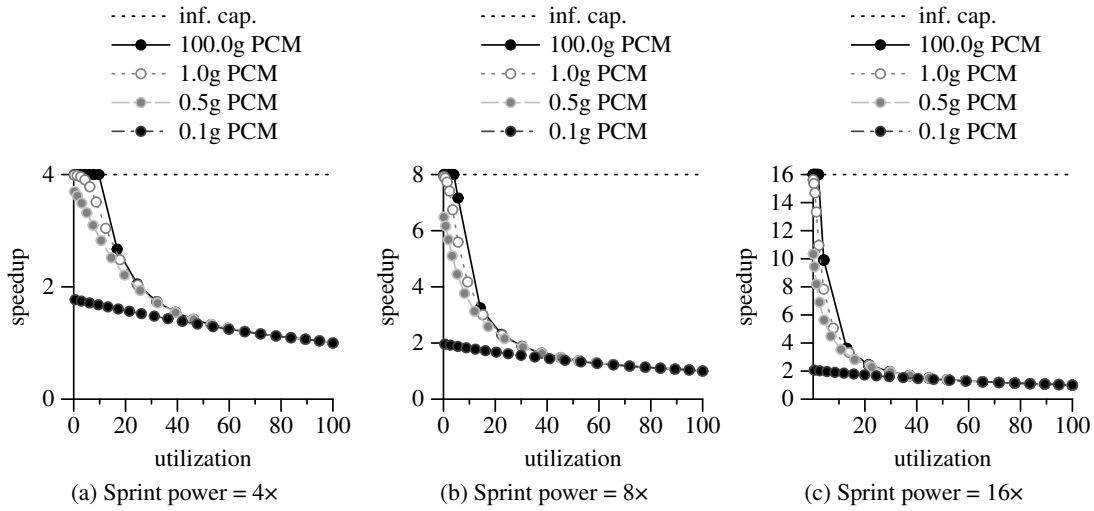


Figure 4.12: Speedup relative to a non-sprinting baseline with varying system utilization for different thermal capacitance. Task size is held constant.

limited by TDP, for a a sprint intensity of $N\times$, speedup expectedly decreases as utilization begins to approach a factor of TDP/N.

Hence, thermal capacitance (and consequently sprinting) only improves the performance of a system that is relatively underutilized. However, when such instances of utilization are concentrated within a few bursts of intense activity followed by prolonged idle periods, a *hierarchical* approach to thermal capacitance can allow the system to sprint more effectively, as described next.

4.3.3 Periodic Computation Bursts: The Case for a Thermal Hierarchy

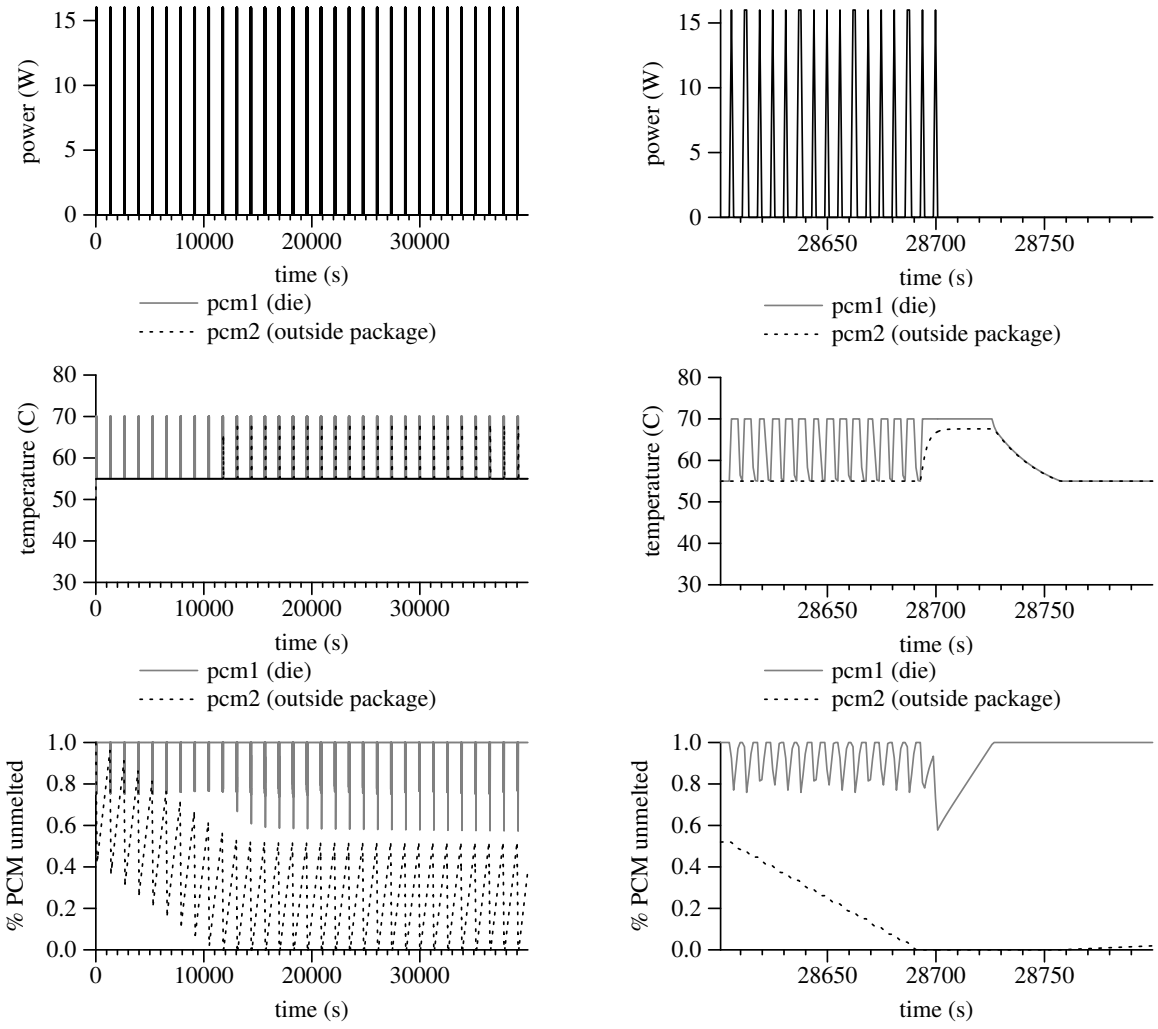
Although the utilization of today's phones is typically low, they are typically distributed across a few "bursty sessions" through the day. For example, one study of 255 users showed that although application preferences varied across the users, the number of "interaction sessions" per day varied between 10 and 200, with the mean interaction lasting between 10 and 250 seconds (these interactions include applications such as calendar and email which may not contribute to computation load) [50]. The study found that devices are seldom used during the night and intermittently used in such bursty interactive sessions during the day with application preferences varying across users. Sprinting would hence be most effective when sufficient thermal headroom is available during these instances of interaction.

Consideration for a thermal hierarchy. Because only a limited thermal mass can be introduced close to the die, one approach is to provision such capacitance to buffer heat during individual sessions, but dissipate this heat to a larger thermal capacitance between sessions. The insight is that although the TDP of the system is ultimately constrained by passive convection, higher thermal conductivity internal to the device allows the die itself to cool quickly enough to sprint repeatedly. The larger thermal capacitance later cools down during prolonged durations when the device is idle. When leveraging phase-change for heat buffering, selecting a higher melting point for the smaller capacitor closer to the die (and a lower melting point for the larger, farther away capacitance) may further enhance heat transfer away from the die.

Example operation of repeated sprints in a thermal hierarchy. The next experiment illustrates the benefits of such a thermal hierarchy in the mobile system considered in Section 4.1. This example associates a small amount of PCM with the die (0.5 g, melting point 70°C) and a larger amount of PCM (5 g, melting point 55°C) placed farther away (for example outside the package, but within the mobile phone case). The sprinting assumptions, TDP of the system and the temperature limits are similar to Section 4.1 (*i.e.*, 16 W sprint power, 1 W TDP, but with initial and max temperatures of 50°C and 75°C respectively—these temperatures reflect the real-system operating conditions from the subsequent chapters). However, the thermal conductivity between the die and external PCM is assumed to be much higher than the TDP limits imposed by passive convection (3 K/W vs 22 K/W); hence heat transfer between the two phase change materials is expected to be faster than between the larger PCM and the ambient environment.

Consider an example scenario of a user session every twenty minutes, where each session lasts for two minutes. Further, consider each session comprising of tasks of a fixed length (this evaluation assumes that each task would execute for twenty seconds on a non-sprinting baseline), separated by think times of 4 s between tasks. The average utilization of the device itself is approximately only 10% across sessions (2 minutes in every 20 minutes); however, within each session, the system is utilized over 80% of the time.

Figure 4.13a shows the operation of the sprinting system under the above usage scenario. As visible in the power profile, the system is able to always sprint with 16 cores in response to the computational demand. The middle and bottom graphs in the figure show the instantaneous thermal state of the system (temperature and available amount of PCM respectively). The long time constants in the thermal cycles of the larger PCM show that the gradual heat transfer facilitates the



(a) Multiple sprints over 10 hour window

(b) A 100s time slice

Figure 4.13: **Repeated sprints using distributed thermal capacitors.** Power (top), temperature (middle) and amount of available PCM for sessions of multiple sprints (Figure 4.13a). Figure 4.13b zooms in on a 100 s window.

effective redistribution of energy across sessions. Figure 4.13b focuses on a hundred second time window after the initial transients have settled. This figure confirms that the smaller PCM provides the immediate thermal headroom required for the high-frequency computation bursts within each session.

The above example shows that although sprinting by definition restricts *average* utilization over long timescales, adapting the thermal design of a system to the expected usage of a device can enable high performance during occasions of necessity.

4.3.4 Further Considerations to Multiple Sprints: Energy-efficiency and Scheduling

Section 4.3.2 assumed a model where energy was proportional to performance. However, as shown in Section 6.3, in practice, a specific mode of operation may be either energy efficient (when saving background power) or energy inefficient (when the overheads of voltage boosting supersede any such savings). The energy characteristics of specific sprinting configurations hence adds a further dimension to sprinting performance.

Although user studies observe that over 90% of interactions on phones deploy only a single application [50], future devices may enable multiple concurrent applications. In such scenarios, system software (such as the operating system) could additionally make scheduling decisions depending on application characteristics and available headroom. An initial approach proposed a predictive model to separate *best-effort* sprints from *guaranteed* sprints based on application QoS (quality of service) requirements [164]. Researchers have also proposed dynamically trading-off performance between CPU cores and GPUs based on application phases and chip local temperatures [129]. Extending such schemes to consider thermal capacitance could help scheduling software to enhance sprinting performance.

4.4 Discussion on Sources of Thermal Capacitance

The approach proposed so far considered an idealized PCM within the thermal envelope of a mobile phone. This section first examines the more conventional source of thermal capacitance—the specific heat of materials commonly incorporated at the package and device level (such as metallic mass within processor packages and mobile phone cases) (Section 4.4.1). Section 4.4.2 then in-

vestigates the feasibility of the suggested straw-man proposal of exploiting the latent heat of phase change by considering the state-of-the-art in engineering PCMs with the desired properties such as melting point, heat spreading, and repeated thermal cycling.

4.4.1 Heat Storage Using Solid Materials

As seen in Section 4.1, the thermal capacitance of the die is too small to support meaningful sprints. That section also alluded to one limitation of leveraging the thermal capacity of the mobile phone case, *i.e.*, the heat conduction from the processor to the case. For the example mobile phone (Table 4.1), the intermediate thermal resistance $R_{package}$ between the package and the case would restrict the maximum sprint power to only 4× beyond TDP (For $R_{package}$ of 12 °C/W and the previously considered temperature swing between 25°C and 75°C, the maximum heat flow is $(75°C - 25°C)/12\ °C/W) = 4.2W$, or 4× over the 1 W thermal design power). For the desired objective of 16 W sprints, the thermal conductivity would therefore need to be 4× higher.

Limited thermal headroom in existing devices. The above estimate is further exacerbated by the temperature constraints of the device itself. Firstly, the rate of heat flow falls if the ambient environment is hotter; for a phone resting in a user's pocket, the initial (ambient) conditions of a phone case are more likely to reflect the hotter body temperature (37°C) than the cooler 25°C room temperature. Secondly, to prevent user discomfort when holding the device, the maximum case temperature is itself limited to only a few degrees above body temperature. A study involving human subjects [19] showed that that for human skin, acceptable temperatures for contact surfaces that are as conductive as aluminum (or similar metallic surfaces) are in the range of 40° C. Thirdly, the outer surface of a device is typically the bottleneck to heat flow (*i.e.*, passive convection for a mobile phone case), and hence cools down slowly. The above observation implies that the thermal mass of surfaces close to the user can be used to buffer heat over a temperature increase not exceeding 2-3°C.

Adding thermal mass from metals. A straightforward approach to increase thermal capacitance available for sprinting is to place a large (relative to the die) piece of copper or aluminum in close proximity to the die. For example, copper has a volumetric heat capacity of 3.45 J/cm^3 K, hence absorbing 16 joules with a 7.2 mm thick block of copper (or a 10.3 mm thick block of alu-

minum, which has a volumetric heat capacity of 2.42 J/cm^3 K) over a 64 mm^2 die will result in a temperature rise of $10°C$.

Such thermal capacitance from metals can enhance the system's ability to sprint, albeit to a limited extent; Chapter 5 illustrates how the heat spreader (225g of copper) found inside a desktop package provides sufficient thermal capacity to allow for a few seconds of sprinting at 5× the sustainable power of the testbed system. Phone chips do not currently employ such internal heat spreaders because designs seek to reduce package weight and thickness. To design for sprinting under these constraints, it is opportune to explore means of increasing thermal capacitance while remaining sensitive to weight, volume and cost.

Increasing thermal capacitance using metals for transient thermal management is not a new approach. For example, prior work has used large thermal capacitance to exceed the nominal TDP of a portable computer by up to a factor of four for an hour using a half kilogram aluminum plate attached to the case [30]. There are two potential drawbacks, however, which might limit the applicability of such an approach for sprinting (*i.e.*, when the metal block is close to the die): (i) when the system initiates a sprint after operating a single core (at TDP) for an extended period, the metal temperature would already be elevated, limiting the potential headroom available for sprinting, and (ii) the thermal resistance within the (comparatively thick) metal would limit the rate at which heat can be absorbed, and thus could limit sprint intensity (similarly, plastics such as those used in printed circuit boards have specific heat in the range of 0.5 to 1 J/gK, but poor heat spreading in the range of 0.3 W/mK, and hence cannot be effectively exploited for heat storage).

Limitations of sprinting with available thermal mass. Hence, in existing designs, limited thermal capacitance close to the die limits the intensity and duration of sprinting. Commercially sold processors have already demonstrated temporary power-boosts beyond TDP by leveraging external thermal capacitance; however, the intensity of sprinting is limited because of the thermal resistance between the processor and far-away heat storage. (For example, Intel's TurboBoost 2.0 exceeds TDP by 25% for several 10s of seconds). More intense sprinting over meaningfully long durations, hence calls for larger thermal capacitance close to the die.

4.4.2 Heat Storage Using Phase Change

An alternative source of thermal capacitance is the latent heat in a phase change material. As seen in Chapter 2, latent heat refers to the heat required to transition a material from one phase to another (*e.g.*, melting a solid). Such materials add heat storage from not only sensible (specific) heat, when the temperature rises to the melting point, but largely due to the latent heat when the phase-transition occurs. Previous research has proposed PCMs with thermal transients of tens of minutes to hours for thermal management of mobile phones [161] and other portable electronic devices [11]. Most such previous works focused on utilizing thermal capacitance over minute to hour time scales [68, 82, 148]. However, a thermal solution with PCMs that melt within short, near-second timescales and can be situated close to the die presents unique challenges in the choice of a material with suitable properties, but also in integrating it within mobile time constraints.

Melting point. The melting point of the PCM needs to be low enough so it melts when the system is sprinting, *i.e.* $\leq T_{jmax}$. However, to efficiently utilize the capacitance only when sprinting, the melting point should be high enough that the PCM does not melt when the system is not sprinting (*i.e.*, at nominal temperature of the die when it is performing routine background tasks). Typical operating ranges are in the range of T_{jmax} at 75 $^{\circ}C$ to nominal temperature of 50 $^{\circ}C$.

A wide variety of nonflammable, non-corrosive PCMs exist that have phase-transition temperatures in the above range. Paraffins with melting points between 35$^{\circ}C$ and 70$^{\circ}C$ have previously been used in applications ranging from cladding in building walls (to melt during the day and refreeze at night) to increasing thermal conductivity in heat sinks [77, 155, 181]. Thermal greases with melting points within typical chip operating temperatures are routinely used to fill contact gaps. Although these materials have not been explicitly engineered for sprinting timescales, Chapter 5 shows that they can be made to melt while sprinting with a paraffin infused heat sink. Further, the melting point of PCMs can be explicitly engineered by blending polyolefins and paraffin (which have different melting points) [119]. Such blended PCMs have even been used in commerically sold heatsinks, although they are intended for heatsinking over larger timescales (http://www.intermark-usa.com/products/Thermal/index.shtml).

Latent heat. For most substances, the latent heat absorbed during phase-change is significantly larger (5-10×) than the sensible (specific) heat absorbed while the temperature rises to this point [143]. Icosane (candle wax) for example, has a large latent heat of 241 J/g [11]—melting a

single gram of wax would absorb over 200 J. In contrast, over 50g—or 50× more mass—of copper would be required to store the same amount of heat over a $10°C$ increase in temperature. Even accounting for the lower density of PCMs (density of paraffin is roughly 1 g/cc, whereas copper is 8.9 g/cc), the volume, and hence thickness of PCM on a chip of the same area, needs to be less than a fifth that of copper. This relative size and weight advantage of PCMs make them more attractive especially within the tight volumetric constraints of mobile devices.

Heat spreading within PCM and encapsulation. Because most PCMs have low thermal conductivity, a heat spreading network must be integrated to achieve the high rates of heat transfer required for sprinting. Prior approaches increase conductivity of PCMs using one of the four following approaches to increase surface area for heat transfer [143, 167]:

- Using fins to increase surface area: when lining heat-sink fins and heat carrying tubes found in desktop and laptops systems with paraffin, PCMs have been shown to melt over timescales of a few minutes [77].

- Immersing metal matrices with high thermal conductivity: metal or diamond microchannels can achieve PCM-based heat sinking on a timescale of 100 μs [64], and fiber mesh carriers [35] could potentially be coated with or composed of copper to improve heat transfer. In addition to enhancing thermal conductivity, such an integrated mesh could improve the mechanical robustness of the PCM to avoid wearout effects such as the formation of cracks or voids that might compromise thermal conductivity [35].

- Dispersing high-conductivity particles: researchers have found that inserting metal spheres or graphite into paraffin can increase heat transfer by up to 3× [49, 145]. More sophisticated techniques like chemical vapour deposition have also been explored to greatly increase heat transfer within PCMs while retaining high thermal capacitance [17].

- Increasing surface are of the encapsulating container: solid-liquid phase-change materials require enclosures to retain form upon re-solidification. Besides encapsulated solid-liquid PCMs, solid-solid organic PCMs offer reduced complexity as the fluid phase is either not present or is self-contained [119, 176]. Proposals with micro- and nano-scale encapsulation seek to improve heat transfer by increasing the area-to-volume ratio [167]. Although these

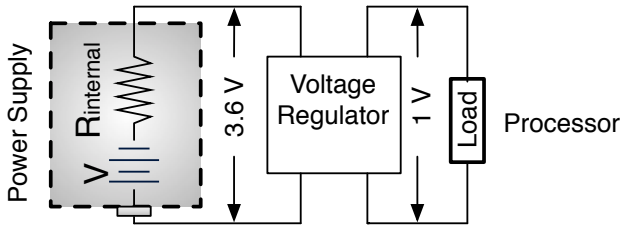


Figure 4.14: Power supply and voltage regulator.

approaches have shown initial promise, key challenges lie in translating them to larger-scale practical applications.

Cost. Engineering a suitable PCM with a suitable heat-spreading network will likely incur additional manufacturing cost. Further, introducing the material close to the die is likely to manifest in added packaging costs. Because much of this work is still exploratory, it is difficult to quantify this cost impact.

Reliability impact of phase-change materials. PCMs are subject to wear-out after repeated cycling. Because existing PCM deployments target large timescales, reliability studies so far have focused on tens of thousands of cycles [68, 82, 148, 155, 161]. Sprinting, however, calls for reliable operation potentially over millions of cycles. Further research is hence required in engineering a material that combines the above properties with higher reliability guarantees. Additionally, the phase-change temperature can trade-off reliability of the PCM with that of the processor. Low temperatures can trigger more frequent phase-cycling in the PCM, but average lower temperatures reduce aging in processors. In contrast, higher phase-change temperatures could result in the processor running hotter on average, while the PCM itself cycles less frequently.

In summary, although PCM-based heatsinks have so far not been designed explicitly for sprinting, the availability of suitable materials and the abundance of prior and ongoing research dedicated to heat transfer and encapsulation of such materials suggest that it may be possible to engineer a thermal solution suitable for sprinting.

4.5 Supplying Peak Power

Another challenge imposed by sprinting is the electrical aspect of supplying and delivering sufficient peak-power in mobile devices. Conventional power supplies and delivery networks are designed

to supply sustainable power to the processor at a steady rate (current) and level (voltage). The additional power required for sprinting increases the demand not only on the power supply, which must now discharge at a faster rate to supply higher current, but also on voltage regulation and power delivery networks, which must continue to keep on-chip voltage levels stable over a wide range of operating currents.

For a typical mobile platform, like the 1 W baseline used in this feasibility study (Figure 4.14), a lithium-ion battery supplies power at a constant 3.6 V, which is stepped down to the processor's rated supply voltage of 1 V using a voltage regulator. The supply current of 1 A (operating power of 1 W at 1 V implies 1 A of supply current) is supplied to the package through I/O pins. Sprinting with 16 such cores on this system would require (i) that the supply can discharge at the rate of 16 A and (ii) that the on-chip voltage rails remain at 1 V despite the large (16×) inrush of current.

In addition to high discharge rate, the power supply also needs to preserve existing desirable characteristics for mobile devices, such as form-factor and energy capacity (battery life). Section 4.5.1 identifies the limitations of using existing battery based power supplies for sprinting. Section 4.5.2 and Section 4.5.3 suggest alternative battery and ultracapacitor based supplies that can enhance sprint intensity. Section 4.5.4 and Section 4.5.5 then address voltage regulation, on-chip power distribution and voltage stability.

4.5.1 Conventional Batteries in Mobile Devices

Portable electronics, including phones and laptops predominantly use Li-ion based batteries as their sole power supply. The chemistry and size of the battery decide its energy capacity at discharge rate. Mobile phone batteries are designed to discharge at most a few amperes of current; the discussion below cites the limitations of using such existing battery technologies to sprint with tens of watts.

A basic Li-ion battery consists of an anode made of porous carbon (usually graphite), a cathode which is a metallic oxide of lithium (usually one of lithium cobalt oxide, lithium manganese oxide, or lithium iron phosphate), and an electrolyte which conducts lithium ions between the two electrodes when the circuit is closed. Newer devices employ lithium polymer batteries which replace the electrolyte in a lithium ion battery with a polymer-based electrolyte. The key advantage is that Li-polymer batteries do not require a hard case, and can instead be manufactured in laminated sheets ("pouch cells"), which allow for thin batteries to be molded into convenient form factors.

Operational parameters: energy capacity, C-rate, output voltage, and self-discharge. Battery capacities are expressed as ampere-hours (Ah) or milliampere-hours (mAh). Mobile phone batteries today are typically rated between 1500 mAh and 3000 mAh. These ratings also capture the intended discharge current (called charge rate, or more commonly C-rate). For example, for a 1500 mAh battery, a discharge rate of 1C is 1500 mA, and a discharge rate of 2C is 3000 mA. The typical output voltage of a fully-charged Li-cobalt battery is 4 V, which falls to approximately 3 V when almost all the charge is drained [111]. The advantage of such batteries is that the output voltage is a largely flat 3.6 V for a large part of battery operation (between 20 and 90% charge). Further, Li-ion batteries have excellent charge retention (low self-discharge rate)—when left unused, these batteries retain up to 80% of the charge for years.

For typical Lithium ion (Li-cobalt) batteries rated at 2700mAh [28], operation at 1C rate can therefore produce up to 10 W of power (2.7A × 3.6V), which is ample for non-sprinting operation (1 V, 1 A), and even allows for some limited sprinting. However, it is less than the 16 W/16 A demand of sprinting with 16 cores. The most direct attempt to increase power draw would be to discharge the battery at a higher rate. Unfortunately, discharging Li-ion batteries at high-C rates is a heavily discouraged practice for reasons discussed next.

Limitations of discharge rate. Although under ideal conditions the total battery capacity and output voltage should both remain invariant of discharge rate, in practice, both output metrics deteriorate as load current increases. Firstly, the rate of electrochemical reaction inside the battery, which determines the rate of charge production, is limited by battery chemistry and form-factor (mobility of reactant ions and surface area available for reaction). Secondly, the effect of internal resistance in the battery, such as from material contacts, is magnified at high current. For the example 16 W sprint in this chapter: (i) 16× current would cause a 16× increase in the voltage drop across the internal resistance, and hence present a correspondingly lower output voltage and (ii) the increased ohmic (I^2R) losses would not only decrease the efficiency of the battery but also increase battery temperature, which can lead to overheating and thermal stress [28].

Battery manufacturers therefore do not recommend operating at high C-rates. In practice, the battery is most efficient at low discharge rates—even lower than 1C. Previous work has suggested that deliberately running the processor at low power—even at the loss of performance—may result in net energy savings due to the in thatcreased energy efficiency of the battery itself [111]. Sprinting in fact suggests that running faster to allow the system to idle sooner is an opportunity to save energy

by allowing "uncore" components to idle sooner (Chapter 6); it is hence undesirable to offset such energy savings with energy losses due to higher C-rate.

Therefore, although overdriving existing batteries may in practice facilitate sprinting, the above drawbacks suggest considering more efficient, high-current power supplies. Devices capable of faster responsiveness may in turn trigger further user activity, ultimately requiring higher energy storage. The growing need for high energy density, high power density supplies for applications ranging from portable electronics to hybrid electric vehicles was raised in a United States Department of Energy report calling for basic research in electrical energy storage [125]. The examination of alternate energy supplies for sprinting below includes some of the resulting research.

Alternative Battery Configurations

A straightforward approach to increase both energy capacity and power capacity combines multiple batteries in series-parallel packs. High-end phone users already employ such packs to extend battery life. However, custom attachments add to weight, size, and cost. Alternatively, sprinting could leverage battery constructions which specifically target high power density by increasing surface area and/or decreasing internal resistance.

Lithium-manganese batteries are capable of much higher power density than their lithium-cobalt counterparts. Such batteries are hence used in high-power applications such as power tools and hybrid electric vehicles. These Li-manganese batteries can supply continuous peak currents of 20-30 amperes and one-second pulses as high as 50 A within a form-factor comparable to Li-cobalt batteries. However, their energy density is over 30% lower (1100-1500 mAH). In contrast, Li-iron phosphate batteries have low internal resistance and hence high current ratings (approximately 100A), but suffer from leakage due to high self-discharge rates.

The Li-polymer batteries mentioned earlier can also provide high peak power density because they are malleable, and can be constructed with large surface area. For example, a representative 51g Li Polymer battery, such as Dualsky GT 850 2s, can supply 43A at 7V. However, they are prone to expansion on overcharging and hence require charge-protection circuits for safety. Today, the relatively high manufacturing costs restrict the use of such batteries to tablet computers and the newest high-end phones.

More recent research demonstrates orders of magnitude improvements in battery power density in nano- and micro-scale constructions. Kang and Ceder [83] note that while existing technology has focused on engineering electrolytes that can migrate electrons fast, the mobility of the Li-ion is itself a bottleneck; coating the ion-conducting surface increases conductivity, resulting in over 50× increase in power density. More recently, Pikul *et al.* [131] constructed a microbattery structure that leverages the energy efficiency of Li-ion batteries, but is able to obtain 2000× higher power density by reducing internal resistance with highly porous 3D electrodes. Although these technologies are yet unproven "at scale", the demand for increasing compute in mobile devices has fueled significant attention towards constructing batteries that combine high energy density and high power density [125].

4.5.2 Ultracapacitors

Ultracapacitors, contrary to batteries, offer high power densities by packing a large amount of energy in the electric field between a double layer of porous carbon [146] coated on a pair of electrode plates. Because charge is directly available—instead of resulting from electrochemical reactions—ultracapacitors do not suffer from the discharge rate limitations of batteries. However, ultracapacitors present trade-offs due to their construction and principle of operation: (i) batteries offer significant energy density because the charge is stored in their bulk; ultracapacitors, in contrast, only store charge in the surface (albeit a large surface area inhabited by micro-pores) and hence offer an order of magnitude lower energy density than batteries of the same size (note that the energy density of ultracapcitors is still several orders of magnitude larger than traditional electrolytic capacitors), (ii) although ultracapacitors can be discharged extremely quickly, the voltage drops linearly when discharging, making it a challenging task to use ultracapacitors as the sole source of power, and (iii) ultracapacitors typically suffer from higher self-discharge rates than batteries.

In the effort to close the energy density gap between batteries and ultracapacitors, recent research has resulted in proposals to greatly increase the surface area available to hold charge. Replacing porous carbon with "forests" of carbon nanotubes increases surface area for charge adsorption, and promises up to 50% the energy density of an equivalent sized battery [146]. Laboratory prototypes of graphene-based ultracapacitors [159] leverage the large surface area and excellent electrical conductivity of graphene to further bridge the energy density gap.

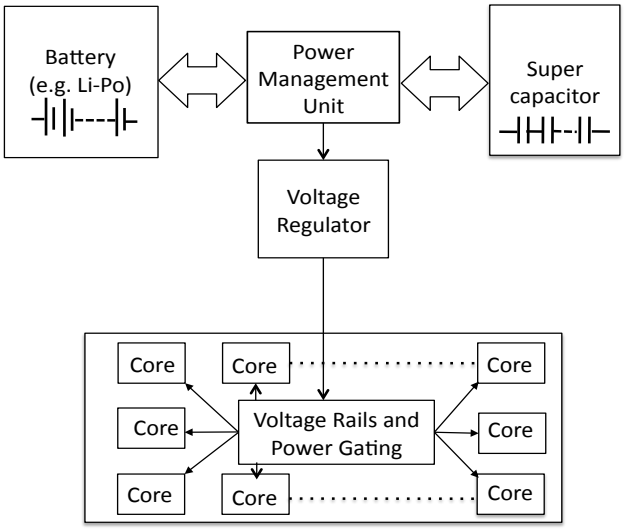


Figure 4.15: A potential hybrid power supply for a sprinting system

Despite these drawbacks, the latest ultra-capacitors easily meet the energy and peak current requirements of *a single sprint* within mobile form factor and low charge leakage constraints. For example, a 25F NESSCAP ultra-capacitor, weighing 6.5g, can store 182 joules and provide a peak current of 20A at a rated voltage of 2.7V with a total leakage current below 0.1mA. By discharging the battery slowly to pre-charge the ultracapacitor and discharging a burst of high-power current during sprinting, a hybrid approach for a sprinting power supply could hence combine the power-density of ultracapacitors with the energy-density of batteries. Such practical battery-ultracapacitor hybrid supplies are already prevalent in pulsed-current applications today [96].

4.5.3 Hybrid Energy Storage Systems

Li-ion capacitors are constructed with one electrode (positive) with activated carbon (similar to ultracapacitors), and the other (negative) with graphite (similar to Li-ion batteries). These devices combine the energy-density and low leakage of batteries with the C-rate, low internal resistance and repeated charge-discharge cycles of ultracapacitors [153] Commercial Li-ion capacitors like the JSR Micro ULTIMO are available in thin light-weight pouches (180 x 126 x 5.5mm), can store 1100F of charge with an output voltage between 3.8 and 2.2 V (for a total energy between 3 - 8 KJ) with less than 5% self-discharge over 3 months, and exhibit negligible loss in capacity even at 10C discharge. The main drawback of this technology, however, is that although the energy density is 2-

3× higher than ultracapacitors, it is still 8-10× less than regular Li-ion batteries (36 KJ for a regular 2.7mAH Li-ion battery)

Prior work has explored hybrid battery–ultra-capacitor power sources to support burst currents in larger electronic systems [130] and to improve battery efficiency [117, 126]. Hybrid energy systems trickle charge from the battery to the ultracapacitor which can then be used for fast-discharge (high power) applications; the key advantage of such systems lies in avoiding energy-inefficient, high-C battery discharge. Inspired by hierarchical memory systems (where storage differs based on latency or capacity), such hybrid electric systems propose incorporating heterogeneous power supply banks with varying energy density, power density and self-discharge characteristics. To exploit such systems effectively, the individual characteristics of supply elements is matched to the load. For example, holding the charge in ultracapacitors ensures high power availability during periods of high activity, but it can be wasteful during idle periods due to the relatively higher self-discharge rate. Building on the memory-hierarchy analogy, researchers have proposed mechanisms for charge allocation, migration, and replacement for optimal usage [130, 174, 179].

A hybrid supply like Figure 4.15 can potentially supply sufficient power for sprinting while retaining high energy capacity. Previous work suggests the feasibility of such an approach: hybrid systems have been successfully in applications to improve battery runtime [126], store solar energy [59], and power vehicles. Such hybrids have also been demonstrated in pulse-operated power systems, showing that two 100F ultracapacitors coupled with two regular 18650 lithium-ion cells achieved 132W peak power in millisecond bursts [56]. However, the additional components and power management will likely increase cost and design complexity.

Overall, the above approaches suggest that an ultracapacitor-battery hybrid weighing only a few grams could be an attractive power supply to support a few seconds of sprinting with tens of watts.

4.5.4 Voltage Regulation and Supply Pins

As mentioned earlier, a voltage regulator is used to supply a stead, operating voltage level to the chip. Consequently, the voltage regulator is responsible for stepping down the output voltage of the supply to the chip's operating voltage, and further keeping the voltage constant in response to fluctuating load. Sprinting may influence the choice of voltage regulators because (i) the load

current can vary over a wide range (1 A in baseline operation to 16 A when sprinting at full intensity) and (ii) the supply voltage may additionally vary if ultracapacitors are discharged.

The task of stepping down voltage is accomplished using linear or switching regulators. Linear regulators employ a voltage divider resistance ladder, which can be easily fabricated at small sizes. However the conversion efficiency (amount of power transferred from source to load) decreases with the ratio of input and output voltages; for example linear regulators can achieve over 90% efficiency when stepping down 1.1 V to 1 V, but less than 28% efficiency when stepping down 3.6 V to 1 V. Switching regulators in contrast sample the input voltage at a duty cycle based on the ratio of output and input voltage, using inductors to store energy. Because their efficiency is less sensitive to the ratio of input and output voltage, switching regulators are deployed in processors which operate over a range of output voltages (most modern processors implement dynamic voltage frequency scaling).

In today's low-power mobile phone processors, the range of load current is relatively small. A sprinting system may therefore require wider dynamic range voltage regulators than typical mobile systems. Such regulators have been routinely deployed in desktop and laptop systems. Laptop and desktop processors typically employ multiple active and idle power modes ranging between several tens of watts to over 100 W (in high-end processors) [9, 140].

Both mobile and desktop systems have until recently placed the voltage regulator off-chip (on the PCB) because switching regulators require inductors [90]. However, recent advances have allowed access to smaller inductors on-chip, enabling on-chip voltage regulators in the most recently shipping processors (Intel's Haswell line of chips). Such on-chip voltage regulators allow fine-grained voltage control at the core [90] or even functional unit level [81] and can hence enable smooth transitions into and out of sprint mode.

A further challenge lies in delivering the necessary peak currents from the off-chip power source over the chip pins. Whereas 100A peak currents are commonly sustained in desktop and server package/socket designs, 16A peak currents exceed the norm for mobile devices. Providing such peak currents will likely require more power and ground pins. Today's phone and tablet processors have smaller packages and narrower pin pitches than desktop chips (for example, Apple's A4 has a 14mm-square package, 0.5mm pitch and 531 pins; the Qualcomm MSM8660 has a 14mm-square package with a 0.4mm pitch and 976 pins). If a pair of power/ground pins provides a peak current of 100mA, 16A at 1V requires 320 pins, likely increasing the cost of the package. On-chip voltage

regulators [90] could allow higher input voltages per pin, and hence potentially reduce the number of power and ground pins.

4.5.5 On-chip Voltage Stability

Sprinting also introduces electrical challenges in its on-chip power distribution grid. When transitioning into or out of sprint mode, on-chip voltage rails must remain within specified tolerance levels to preserve state and prevent timing errors despite the in-rush of current when activating many cores [88]. The system must additionally activate cores quickly enough to minimize the delay before useful parallel computation can begin. Prior work has examined activation schedules for large blocks within monolithic cores [80], current staggering techniques to slowly ramp up/down units when activating/deactivating cores [89], and gradual activation of individual cores [137]. These works find core activation/deactivation latencies of at most 100s of microseconds, which is consistent with core transition latencies on modern manycore chips [140, 163]. In comparison with the near-second sprints considered in this dissertation, the electrical transition latency contributes less than 0.001% as an overhead to total sprinting time (see sidebar on *Mitigating Voltage Fluctuation*).

4.6 Hardware-Software Interface for Sprinting

Semantically, sprinting is no different from conventional execution because it only affects performance. However, sprinting beyond the system's thermal capacity can cause the system to overheat. Hardware must therefore detect impending overheating, after which the system must return to sustainable performance by throttling frequency and cores (either by hardware alone or with operating system assistance). This section outlines basic mechanisms for a sprinting hardware and runtime.

Functional requirements of a sprinting system. Software activates parallel sprinting whenever there is sufficient thread-level parallelism in the application. For example, when there are more active threads than cores, the operating system or runtime informs the hardware to wake-up idle cores and migrates threads to the newly activated cores. As seen in Section 1.2, while computing in parallel, the power dissipation of the system exceeds TDP. Because continued activity at this rate eventually leads to overheating, sprinting requires mechanisms to determine the thermal state of the system and throttle operating power.

Mitigating Voltage Fluctuation

To understand the voltage fluctuations caused by increased current draw during sprint activation/deactivation, Yixin Luo and our collaborators at the University of Michigan modeled the component, package and board level electrical network using SPICE (Figure 1(a) and Figure 1(b)) [137].

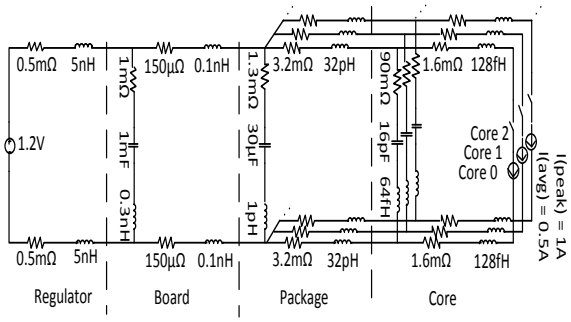
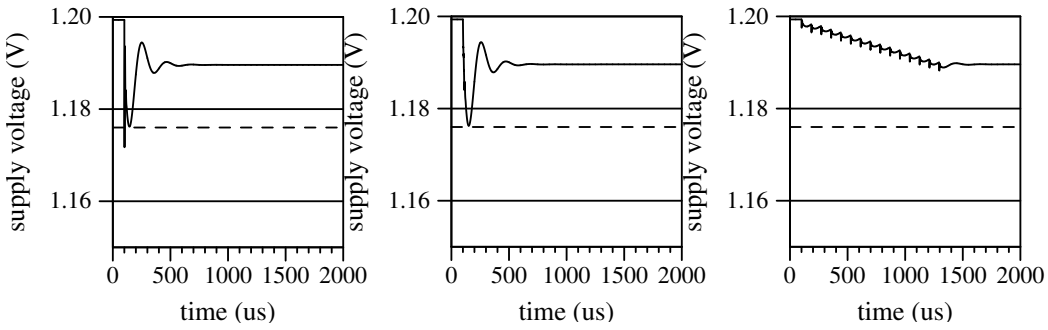


Figure I(a). RLC power network model.

Power source	1.2 V
Regulator	1 mΩ, 10 nH
Board coupling	1 mΩ, 300 pH, 1 mF
Board	300 $\mu\Omega$, 200 pH
Package coupling	1.3 mΩ, 1 pH, 29.6 μF
Package	6.4 mΩ, 64 pH
Core coupling	89.6 mΩ, 64 fH , 15.94 pF
Core	1A current, 3.2 mΩ, 256 fH

Figure I(b). Electrical parameters

The supply voltage waveforms below show that: (a) abruptly activating all cores causes voltage fluctuations to exceed tolerable levels (the dashed line below 1.18V), (b) gradually activating the cores with a delay of 1μs between cores causes the fluctuations to decrease, and (c) slowing down the sequence to 100μs results in safe sprint activation. The time to start/stop sprinting is thus a few orders of magnitude removed from being a performance bottleneck when desired response times are in the range of one second. Modern multicore chips supporting deep-sleep states report similar wake-up latencies [163], [140].



(a) Instantaneous (b) Linear ramp over 1.28μs (c) Linear ramp over 128μs

Figure II. Supply voltage versus time for activating cores over three ramp-up times

Thermal monitoring. As seen in Section 3.4, the thermal state of a system is a combination of: (i) temperature, when heat is absorbed by the *specific heat* capacity of materials and (ii) amount of material left to undergo phase-change, when the system is at the phase-change temperature. Because the threshold limit is specified as a maximum temperature, a straightforward temperature monitor can detect when sprinting needs to be terminated—if temperature sensors provide sufficient resolution. For example, on die thermal sensors in the testbed system (see Chapter 5) register temperatures to 1°C of precision approximately every second. Sprinting at smaller timescales requires either higher resolution (sampling rate) temperature sensors or alternative mechanisms to estimate thermal state.

One such alternative is to estimate temperature based on input power. Given a thermal model of the system, and the operating power sampled during execution, the equations described in Chapter 3 compute the thermal state (temperature, and phase-change if applicable) of the system. Whereas chips such as the IBM Power 7 [175] and Intel's Sandy Bridge [9] provide architectural support for energy/power metering, instruction activity has previously been used successfully to derive operating power in the absence of such support [74]. Systems today employ similar activity-based dynamic thermal management to close the gap between worst-case thermal budgets and average-case power dissipation [29, 151]. For example, today's GPUs keep chip power below TDP by dynamically varying clock frequency based either on direct power metering or on statistics gathered from performance counters. (such as AMD's PowerTune technology).

Truncating sprints to prevent overheating. If all computation completes during a sprint, the operation can end by simply turning of the now-idle cores and placing them into a deep sleep mode. However, when the computation exceeds the sprint capabilities of the system, the sprint must be "truncated" by resuming sustainable operation. In the example case of *parallel* sprinting used in this feasibility study, sprint truncation involves deactivating the 15 additional cores and migrating any active computation to the one remaining core. Although either hardware or software could implement such a mechanism, similar scheduling decisions in today's systems are usually handled by the operating system. For example, on the Sandy Bridge system used in the following chapters, the hardware informs system software of thermal emergencies by setting status bits in a thermal status register. If the thermal emergency persists, the hardware invokes its own throttling measures such as reducing frequency and reducing clock duty cycle [9]. For the case of truncating parallel sprinting, if software is unable to migrate threads and deactivate cores in time, hardware

can similarly throttle frequency of all active cores to remain under sustainable power. As dynamic power dissipation is linearly related to frequency, the hardware must throttle the frequency by at least a factor equal to the number of active cores. Once software has migrated all threads and deactivated cores, computation can resume at nominal clock frequency.

Utilizing thermal capacitance as a computational resource. In addition to the above basic functions to support sprinting, thermal headroom introduces a new computational resource for hardware/software to manage. The goal of a sprinting architecture is to ultimately harness this resource to enhance performance and/or save energy. Chapter 5 and Chapter 6 implement and evaluate functional and energy-performance features of a sprinting runtime.

4.7 Impact of Sprinting on Reliability

Processor reliability has received significant attention in the literature [156]. This section restricts the discussion to how *sprinting* could potentially impact two key factors affecting reliability: (i) current density and (ii) temperature [4].

Sprinting operation draws increased current flow compared to non-sprinting operation. High current density along power rails is known to cause metal ions to drift along the path of electrons, a phenomenon called *electromigration*. Electromigration can cause damage due to voids and cracks, as well as increase heating by causing current to flow along high resistance paths [104]. This phenomenon occurs not only along metal lines, but also on vias and contacts (such as solder bumps on the package). The current density proposed in this work (total source current in the range of tens of amperes over chip areas in the range of 100 mm^2) is comparable to desktop chips today, implying that solutions such as current staggering [89] and careful packaging practices (such as placement of solder bumps) [150] could potentially extend to the context of sprinting operation in future process generations.

The second key aspect of sprinting which impacts reliability—temperature—manifests in commonly accepted failure models as two factors: (i) the mean absolute operating temperature and (ii) thermal cycling [4]. The absolute operating temperature appears as an exponential term in several models of mean time to failure (MTTF) including electromigration (discussed above), stress migration (where differing thermal expansion rates between interfacing materials causes wear out) and dielectric breakdown over time [158]. Although sprinting never drives chip temperature beyond

safe margins, the design of a sprinting system can influence average chip temperature. For example, when incorporating a phase-change material close to the die, a low melting point results in a cooler average chip temperature, which can potentially increase processor reliability. The trade-off behind such a design choice is that the PCM could itself wear-out due to the second type of temperature-related stress—*thermal cycling* [119]. Recent work explores control systems for temperature modulation using phase-change to conserve battery energy by regulating fan speed [173].

In contrast to absolute operating temperature, thermal cycling results from the *temperature delta* during repeated heating or cooling cycles. As mentioned above, such thermal cycles can cause wear-out in the phase-change material itself; the same phenomenon also impacts materials within the processor package. The fatigue due to thermal cycling is known to be most pronounced in the interface between the package and die, such as leads and solder bumps [4, 158]. The commonly understood model for thermal cycling assumes large periods of gradual temperature cycles (a few times daily) and is given by:

$$MTTF_{thermal-cycling} \propto (T - T_{ambient})^{-q}$$

where the exponent q is empirically determined (Coffin-Manson constant, typical value for metals is 2).

For such long-periodicity cycles, for example, resulting from buffering the heat locally for long periods of time (Section 4.3 suggests one approach for such a thermal hierarchy), sprinting does not increase the temperature delta in the above equation. However, the *number* of such cycles depends on future use-cases and specific design constants. In the absence of such a thermal hierarchy, (for example, the thermal solutions evaluated in this feasibility study, and in the real system from the subsequent chapters), sprinting will likely result in *high frequency thermal cycles*—where the temperature transitions occur rapidly over the range of seconds (as opposed to days in the Coffin-Manson model above). As seen in Section 4.1.3, the heat-flux and temperature transient introduced by sprinting are well within the range of modern desktop packages. However, the impact of such high-frequency thermal cycling on integrated chips and packages remains to be comprehensively understood [158].

4.8 Impact of Sprinting on Cost

The proposed approaches to design a system for sprinting seek to provision additional thermal capacitance, enhance power supply via enhanced battery and hybrid-ultracapacitor technologies, and suggest architectures and micro-architectures which are more power hungry (and potentially occupy more area) than conventional mobile packages. Such schemes are likely to increase packaging costs due to added material and manufacturing costs (for example if a PCM is introduced inside the package) and higher pin counts. Further, engineering materials with appropriate thermal properties, form-factors, and energy and power-densities could incur significant investment in research and development. This dissertation merely acknowledges that manufacturing cost could hence be an important factor towards engineering sprinting systems (without any further analysis).

4.9 Chapter Summary

Starting from the projections of dark silicon in future thermally constrained environments, this chapter investigated the feasibility of utilizing tens of reserve "dark cores" to perform short, near-second bursts of computation to improve responsiveness for bursty, interactive applications. Because conventional systems are not designed with sprinting in mind, this feasibility study broadly analyzed the fundamental challenges to such an approach.

4.9.1 Summary of Findings

- When sprinting, a system exceeds its thermal design power. The degree to which a system can sprint depends on the amount of heat which can be temporarily absorbed, *i.e.*, how much thermal capacitance is available close enough to the heat generating die. In today's systems, available thermal capacitance is limited to the die and parts of the package to which heat spreads quickly enough. Augmenting a system with more thermal capacitance close to the die has the potential to further improve sprinting performance (as opposed to sustainable performance at TDP which remains unchanged for a given thermal *resistance*). Analysis using basic thermal models shows that one alternative could be to exploit the large latent heat of phase-change, such as melting a few milligrams of wax.

- How intensely a chip can sprint depends also on the peak current output of the energy source in the system. Conventional phone batteries today are not designed for high current draw; however, the more expensive batteries used in modern tablets, or alternative battery-ultracapacitor hybrid energy sources can provide sufficient power for sprinting with 10s of watts (whereas mobile processors today have sustained power draw of at most 1-2 watts). Staggering core activation (10μs per core), using more sophisticated (laptop-class) voltage regulators, and increasing power and ground pins on the package help to deliver a large range of power (for sprinting, baseline and idle operation) without excessive voltage fluctuations.

- To estimate application performance with sprinting, the feasibility study evaluated performance and energy consumption of a set of vision-based workloads by simulating a future thermally constrained processor. The results showed that parallel sprinting by activating 10s of otherwise idle cores could result in speedups of 10× for these workloads within a 1 W TDP platform. When considering the energy consumption for workload execution, parallel sprinting introduces little to no dynamic energy overheads compared to a non-sprinting baseline.

This chapter is therefore a first step which investigated the feasibility of sprinting as an operating regime by broadly considering the most fundamental technical barriers to sprinting, and identifying engineering choices explicitly focused towards enhancing a system's ability to sprint.

4.9.2 Next Steps

In practice, how sprinting manifests in a device several process generations in the future will likely depend not only on the thermal and electrical properties of future mobile devices, but also on how users and applications adapt to sprinting. Sprinting can therefore benefit from further research spanning user studies, engineering phase-change heatsinks, and leveraging hybrid electric sources. Manufacturing a sprinting chip would also require deeper investigation into packaging costs and reliability issues from thermal cycling.

The developments in battery and ultracapacitor technology promise high power density and energy density sources that would allow significantly higher current draw in future devices. Chips are also beginning to incorporate on-chip voltage regulators and increasingly aggressive power-gating mechanisms to reduce energy consumption. The key remaining engineering challenges likely

lie in mitigating packaging costs due to pin counts, and addressing reliability concerns such as electro-migration from repeated thermal cycling. This dissertation does not further investigate the electrical, reliability or cost aspects of sprinting.

A key observation from the thermal study—that systems can temporarily exceed TDP by leveraging thermal capacitance—has been validated to some degree by the appearance of features such as TurboBoost 2.0 in recent processors. However, the ability to sprint is currently incidental; instead, explicitly designing a system to sprint intensely calls for engineering packages with higher thermal capacitance (such as with form-retentive phase-change materials with high latent heat, appropriate phase-change temperature and high heat spreading). An immediate next step is to leverage the fact that even today's systems have some ability to sprint, because all materials possess thermal capacitance. The next chapter physically validates the feasibility of intense sprinting for a few seconds on a real system today (5× over TDP). This dissertation also examines the practicality of extending sprint durations using a wax-based heatsink (Chapter 5).

From a hardware-software perspective, the key challenge lies in adapting to a system whose performance varies based on its thermal state. For example, a sprint-aware runtime system may need to decide how intensely to sprint based on the nature of workload (*e.g.*, length of a task), the instantaneous temperature, and architectural considerations such as energy overheads when sprinting with DVFS, or performance overheads when parallel sprinting is truncated. The feasibility study evaluated sprinting using simple architectural, thermal and energy models. A more realistic sprinting system involves operating system intervention (such as context switches) which may affect not only parallel performance, but also energy efficiency. Further, energy efficiency on a real system depends not only on dynamic execution power, but comprises of energy from "uncore components" like the cache and interconnect, as well as static (leakage) power. Finally, because processor performance is characterized by both clock frequency and parallelism, sprint policies must choose *how* to sprint based on thermal state and input computation tasks. This dissertation investigates sprinting performance and energy in a full system operating environment that can sprint using either parallelism or DVFS (Chapter 6).

The remainder of this dissertation now focuses on the thermal and energy-performance aspects of computational sprinting on a hardware-software testbed.

Chapter 5

Thermal Response of Sprinting on A Hardware/Software Testbed

The previous chapter used software simulation to analyze the feasibility of sprinting. To further investigate performance and energy of sprinting beyond the accuracy and timescales tractably afforded by software simulation, this chapter introduces an experimental testbed constructed by thermally constraining an off-the-shelf processor. By modifying the cooling solution around a many-core chip so that only its lowest power operating mode (a single core at minimum configurable frequency) is sustainable, the testbed creates an environment that can sprint by activating cores and/or increasing frequency, and is able to execute full-system software including operating system code.

To sprint, a chip must offer an operating point where peak power significantly exceeds sustainable power dissipation (*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 this study. Instead, as a proxy for the thermal characteristics of a future sprint-enabled device, this chapter describes a sprinting testbed system constructed from a multi-core desktop-class chip by reducing its heat venting capacity (TDP) to 10 W (it is originally 95 W) so that only a single core at lowest frequency (1.6 GHz) is sustainable, but the chip can sprint for by up to 5× the sustainable power by activating three additional cores and doubling frequency. The testbed is able to sprint for a few seconds by leveraging the thermal capacitance of the copper heat spreader inside the processor package. To test the hypothesis that phase-change materials can extend sprint duration, this chapter

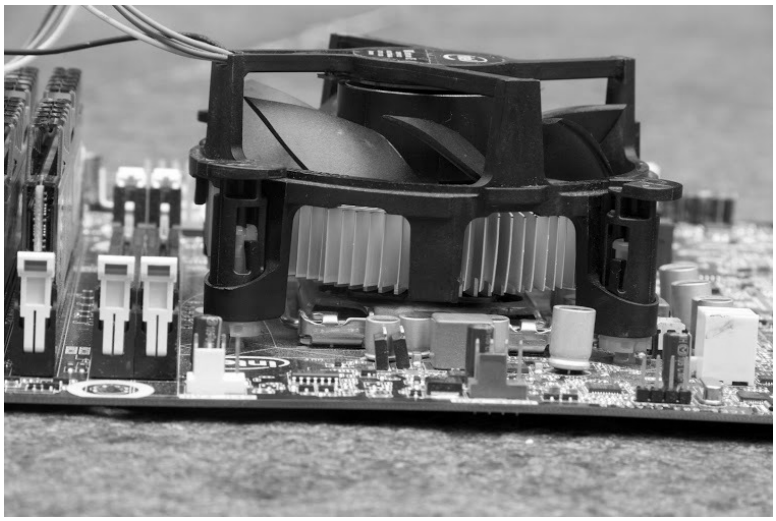


Figure 5.1: Baseline hardware setup with heat sink and fan.

---

also presents the thermal response of the testbed when a small amount of paraffin is placed on top of the package.

Modern desktop chips have well-documented support which allows software to control power states and monitor chip energy and temperature. Avoiding simulation also overcomes modeling artifacts and simulation time constraints [114], such as the simplified energy model and negligence of operating system effects in the simulator used in Chapter 4.

However, the testbed is imperfect in that this study is limited to existing chips/platforms, which have not been designed with sprinting in mind. Further, the power supply and delivery system (including the motherboard) are equipped to sustainably supply the peak-power demands of the chip. The testbed is therefore not meant to faithfully represent a future mobile device; rather, it represents a concrete energy-performance point which nevertheless approximates the gap between peak and sustainable power projected in future thermally constrained devices.

This chapter characterizes sprinting operation on the testbed, focusing on the thermal response. The next chapter uses the testbed to evaluate the performance and energy aspects of sprinting.

5.1 Unmodified System

To create an environment for sprinting, this section first identifies and characterizes the active power modes using controls available on a commodity desktop processor. Then, by restricting heat dissipation to the lowest active power mode, all unsustainable higher power modes can be used only temporarily for sprinting.

5.1.1 Configuration and Monitoring

The baseline platform consists of a quad-core Intel i7-2600 "Sandybridge" processor, with a thermal design power of 95 W. The power supply and distribution network on the motherboard are designed to carry sufficient power for the chip to reliably operate at peak power. To support the rated heat dissipation, by default the processor socket is mounted with a finned aluminum heat sink and a variable speed, software-controllable fan. Figure 5.1 shows this baseline setup.

Power mode configuration. The processor chip consists of four cores which share a 8 MB L3 cache, and also contains an on-die graphics unit. When all four cores are active, the maximum rated frequency (of each core) is 3.4 GHz under default cooling conditions. The idle and active power states of the cores can be controlled by software (through the Linux ACPI interface [1]). Due to the shared voltage domain on this chip, each active core operates at the same configurable frequency, although individual cores can be put into idle modes.

The processor also implements Intel's TurboBoost 2.0 technology, which allows for frequency and voltage boosting beyond 3.4 GHz [141]. However, due to the lack of sufficient software controls, TurboBoost is disabled for the experiments in this dissertation; frequency is instead explicitly controlled using userspace governors. Similarly, to reduce experimental noise, hyperthreading is disabled, the operating system runs minimal services, and graphics/displays are turned off.

Energy and temperature monitoring. The chip allows software monitoring of both energy and temperature through status registers. A package-domain counter (MSR_PKG_ENERGY_STATUS) reflects a hardware-collected aggregate of energy usage calibrated at a fine-grained resolution (set to update with a precision of 15.3 micro-joules every milli-second for this setup). The counter reflects the total package-level energy consumption, including static energy. A simple experiment confirmed that the energy counters were well calibrated: the difference in power between idle and compute intensive loops matched off-the-wall power meter readings—although the meter measures

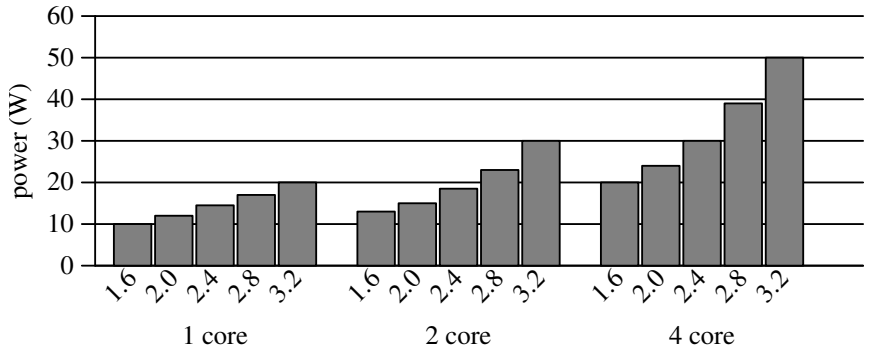


Figure 5.2: Power profile for core and frequency settings.

total system power, the only activity causing the power to increase was due to the computation. The differently measured powers remained consistent across frequency settings.

On-die thermal sensor readings are also available through status registers. On this setup, these registers reflect per-core temperature up to 1° C precision updated approximately every second. For this chip, a temperature reading of 78° C is considered "hot" by the manufacturer [9]; hence, all experiments cap maximum temperature at 75° C to preclude performance throttling by the hardware/operating system. None of the results reported in this dissertation were affected by such external throttling mechanisms.

5.1.2 Processor Power Profile

The above configuration and monitoring facilities allow straightforward characterization of processor power consumption by varying core counts and frequency. Figure 5.2 shows the power consumption when executing the simple, compute intensive `sobel` workload with one, two and four cores by varying the frequency from 1.6 GHz (lowest configurable) to 3.2 GHz in steps of 400 MHz.

As expected, operating power is lowest for the single core at the lowest frequency (10 W at1.6 GHz), and increases with frequency and core count. Running with all four cores active at twice the frequency (3.2 GHz) consumes 50 W. Therefore, this system exhibits a 5× power gap between the least and highest performing modes; correspondingly, the peak expected speedup as a result of activating four times as many cores, and doubling the frequency is 8×.

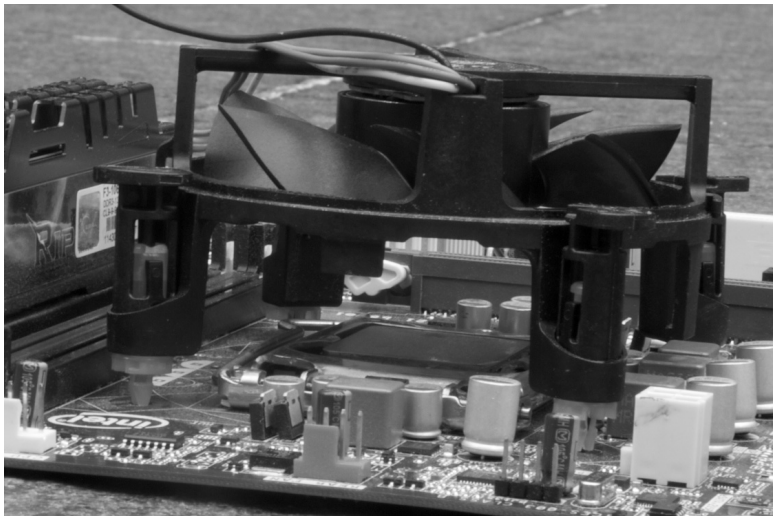


Figure 5.3: Testbed setup

This power-performance gap motivates the approach for creating an environment for sprinting: if the heat dissipation of the system is reduced to ten watts, then this chip can sustain only a single core at 1.6 GHz, but can sprint temporarily by up to a factor 5× in power (and 8× in performance)—comparable, albeit less aggressive than the estimated 10× gap from Chapter 2.

5.2 Constructing a Sprinting Testbed

The baseline system is able to sustain operation at peak power due to its large finned heatsink and fan. A direct approach to establish a single core at 1.6 GHz as the only sustainable operating mode is to replace the heatsink and/or fan to only dissipate 10 W. How long a sprint (by activating additional cores and/or increasing frequency beyond 1.6 GHz) can last then depends on the available thermal capacitance in this augmented testbed. Section 5.2.1 first illustrates the approach to reducing thermal design power. Section 5.2.2 then describes the thermal capacitance of the internal heat spreader found inside the processor package which enables sprinting.

5.2.1 Constraining Heat Dissipation

The goal of augmenting the platform is to reduce its TDP to 10 W from its initial rating of 95 W with both the heatsink and the fan. Simply turning off the fan lowers TDP, but not by enough—

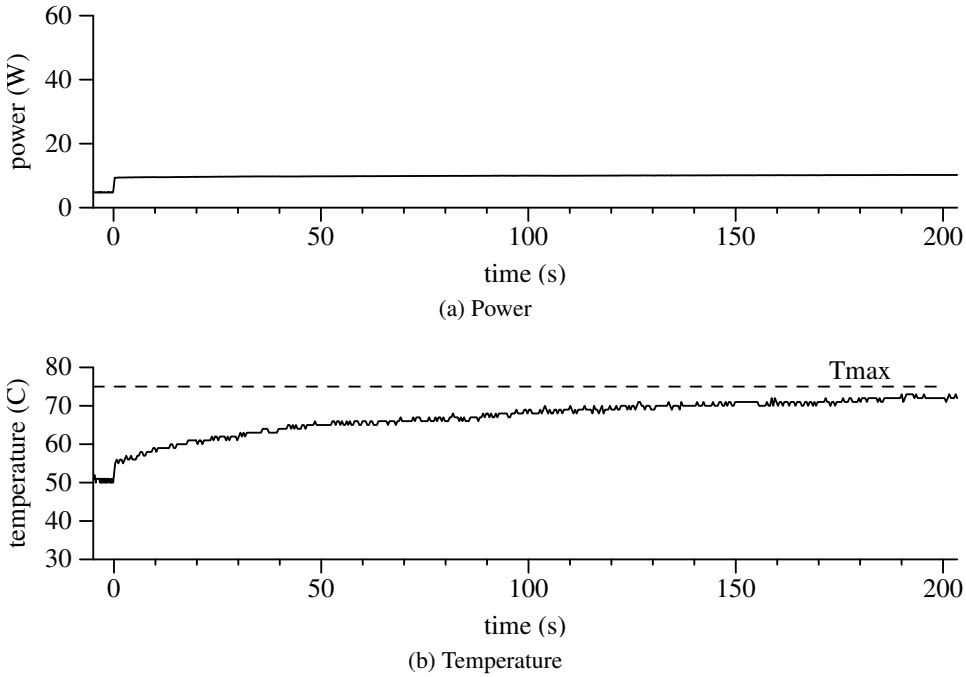


Figure 5.4: Thermal response of sprinting testbed under sustainable execution

the heatsink alone dissipates well over 10 W, allowing higher frequencies (exceeding 1.6 GHz) to be sustainable. In contrast, removing both the heatsink and the fan results in a system that cannot sustain any activity—even at 10 W, die temperatures eventually exceeded the set threshold. However, by removing the heatsink, and selecting an appropriate fan speed (chosen as 1054 RPM after experimentation), die temperature saturates at 75° C (just under the rated junction temperature limit of 78° C).

Figure 5.4b illustrates the thermal response of this augmented system in response to sustained input power (Figure 5.4a). Initially, the system is idle, consuming 4.8 W and at an equilibrium temperature of $50°C$. At time 0, activating a single core at 1.6 GHz causes the power draw to increase to 10 W, and the temperature rises in response. After over 200 s of activity, the temperature is observed to settle at $75°C$. Similarly, Figure 5.5a and Figure 5.5b show the transient response when the processor is returned to idle after sustained operation. At time 0, the power draw drops from the active (10 W) power to idle (5 W), causing the temperature to exponentially decrease from $75°C$. After approximately 5 minutes of idle time, the temperature settles towards the initial value of $50°C$. Careful inspection of the figures also reveals the thermally induced increase in leakage

power: in Figure 5.4a the active power increases from 9.7 W to 10 W as the temperature increases, whereas in Figure 5.5a the idle power drops from 5 W at 75° C to 4.8 W at 50° C.

Thus, for fixed initial and final temperatures (at $50°C$ and $75°C$ respectively), the TDP of this system can now be ascertained as approximately 10 W. By substituting the above values in Equation 3.1, the thermal resistance to ambient, and the ambient temperature can also be computed:

$$T_{eq} = T_{amb} + P \cdot R_{j \cdot a} \tag{5.1}$$

$$\implies 50°C = T_{amb} + P_{idle} \cdot R_{j \cdot a}, \text{and}$$

$$75°C = T_{amb} + P_{sustainable} \cdot R_{j \cdot a} \tag{5.2}$$

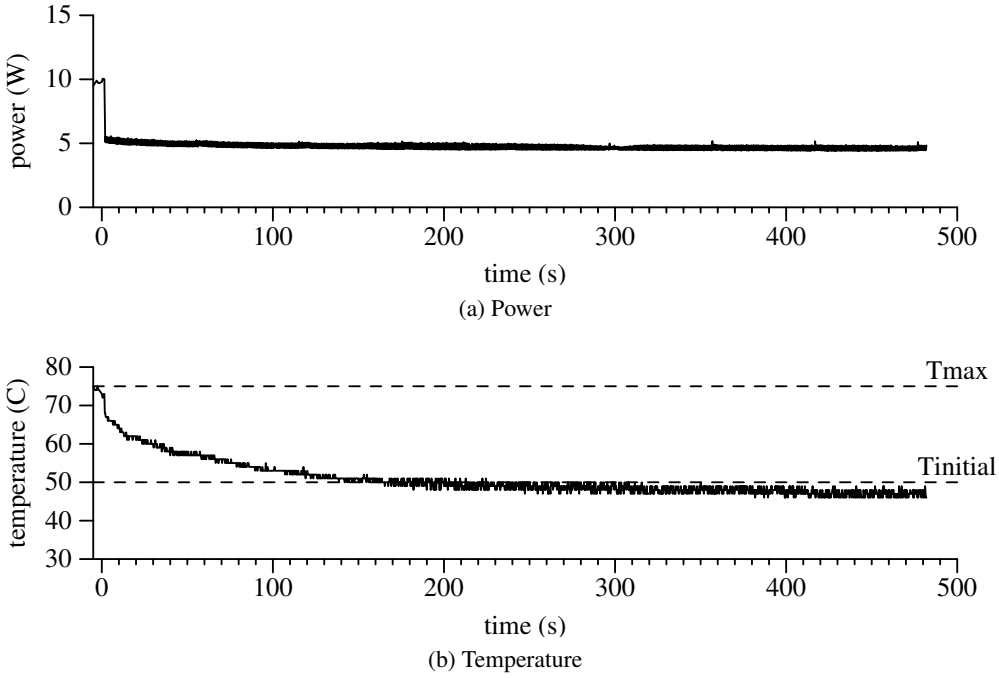
Substituting for P_{idle} as 5 W and $P_{sustainable}$ as 10 W, gives $R_{j \cdot a}$ as $5°C/W$ and T_{amb} as $25°C$, which closely matches the room temperature during the experiment.

When computing with higher frequency or with more than one core, the increased power draw will cause die temperature to rise beyond $75°C$. For example, when sprinting with maximum power (50 W), Equation 3.1 projects an equilibrium temperature of $275°C$. However, these higher power modes can still be used for sprinting; how long a sprint of a given intensity (power) can be active before the temperature increases by $25°C$ (*i.e.*, from the initial to final temperature), depends on the thermal capacitance available on the testbed.

5.2.2 Thermal Capacitance from Internal Heat Spreader

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, contain 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 5.6a. 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.

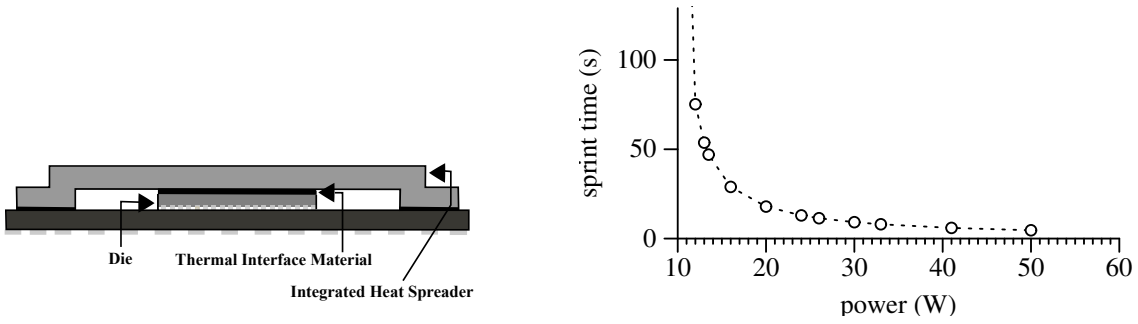
The copper IHS provides sufficient thermal capacitance for sprinting. Given its dimensions (32 mm × 34 mm × 2 mm [8]) and the density (8.94 g/cc) and specific heat of copper (0.385 J/gK), its total heat capacity is 7.5 J/K. When the system idles, its temperature settles at roughly 50° C. Thus, during sprinting, for a temperature swing of 25° C to a maximum temperature of 75° C, the IHS can store 188 J of heat. Figure 5.6b shows the estimated sprint time across the range of



(a) Power

(b) Temperature

Figure 5.5: Thermal response of testbed when idling.

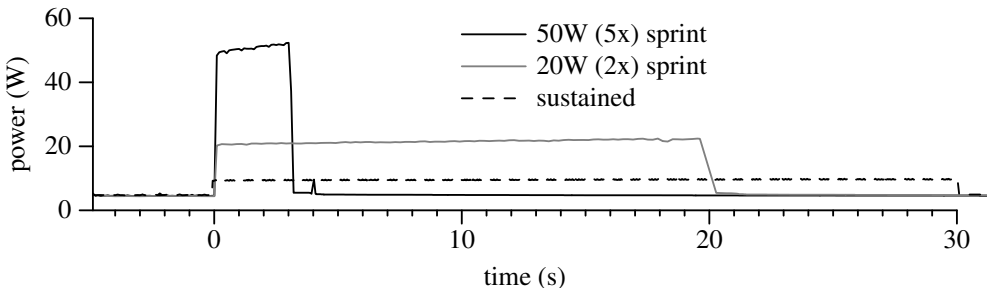


(a) Package internals

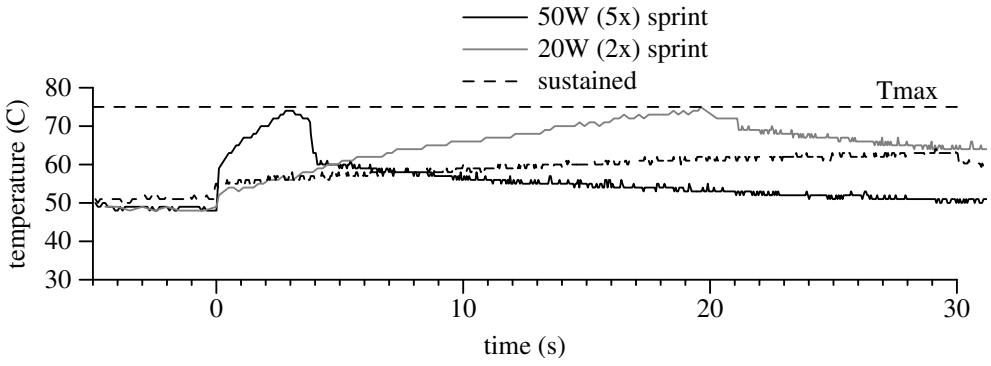
(b) Estimated sprint time

Figure 5.6: Package cut-away showing internal heat spreader (IHS) (Figure 5.6a), and expected sprint duration estimated from thermal capacitance of the heat spreader (Figure 5.6b)

operating power for the testbed system considering that the entire mass of the copper heat spreader is the only thermal capacitance available for sprinting. When the operating power exceeds the sustainable 10 W, the processor can only sprint for a finite duration, which decreases as sprint intensity (operating power) increases. In theory, for the most intense sprint at 40 W over TDP



(a) Power



(b) Temperature

Figure 5.7: Thermal response of sprinting testbed under sustainable execution, sprinting with 50 W and sprinting with 20 W.

(50 W total), the IHS present in this off-the-shelf package can enable 4.7 s of sprinting. The next section compares the above estimates with experimental observations.

5.3 Testbed Power and Thermal Response while Sprinting

Having established a baseline thermal environment, the next step experimentally verifes that the tested is indeed able to sprint. To observe the temperature of the system when sprinting, the processor can essentially be operated as a heater by configuring its operating power.

5.3.1 Sprinting with Maximum Intensity

To determine how long the processor can sprint at its highest observed intensity, the first experiment activates all four cores at maximum frequency from an initial idle equilibrium. Figure 5.7a shows that activating sprinting at time 0 causes an operating power of approximately 50 W. In response,

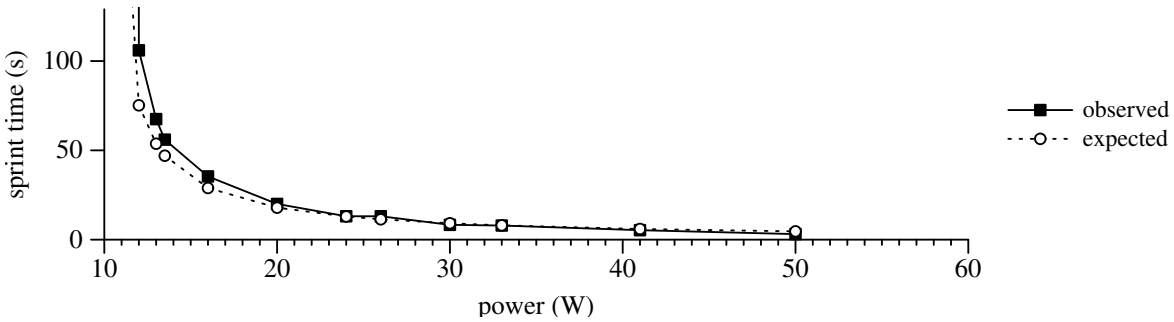


Figure 5.8: Comparison of estimated and observed sprint duration on the testbed.

the chip temperature (Figure 5.7b) increases sharply, and approaches T_{max} in 3.1 s. After sprinting, the processor returns to idle power and chip cools towards the initial temperature matching the previously observed transient (Figure 5.4b).

Compared to sustained operation, the 5× increase in power causes the temperature to expectedly increase much more sharply. The effect of temperature induced increase in leakage power is also more prominent in Figure 5.7a—over the 3.1 s of sprint activity, package power increases from 48 W to 52 W. Finally, the observed sprint duration falls short of the estimated sprint duration, implying that fewer joules heat were absorbed during the sprint than expected; the causes for this divergence are discussed below.

5.3.2 Sprinting with Lower Intensity

As seen in Figure 5.6b, the testbed can potentially sprint for longer durations at lower sprint intensities. Figure 5.7a illustrates a second example mode of sprinting at 2× the sustainable power (*i.e.*, 20 W operating power). The corresponding thermal response (Figure 5.7b shows that the temperature now rises less rapidly compared to the 50 W sprint, and the processor sprints for twenty seconds before temperature approaches T_{max}. The post-sprint phase follows the expected exponential cool-down transient. The observed sprint duration (20 s) now exceeds the estimated sprint duration from Figure 5.6b (16 s), implying that more energy was available for sprinting than expected.

5.3.3 Effect of Non-uniform Thermal Capacitance

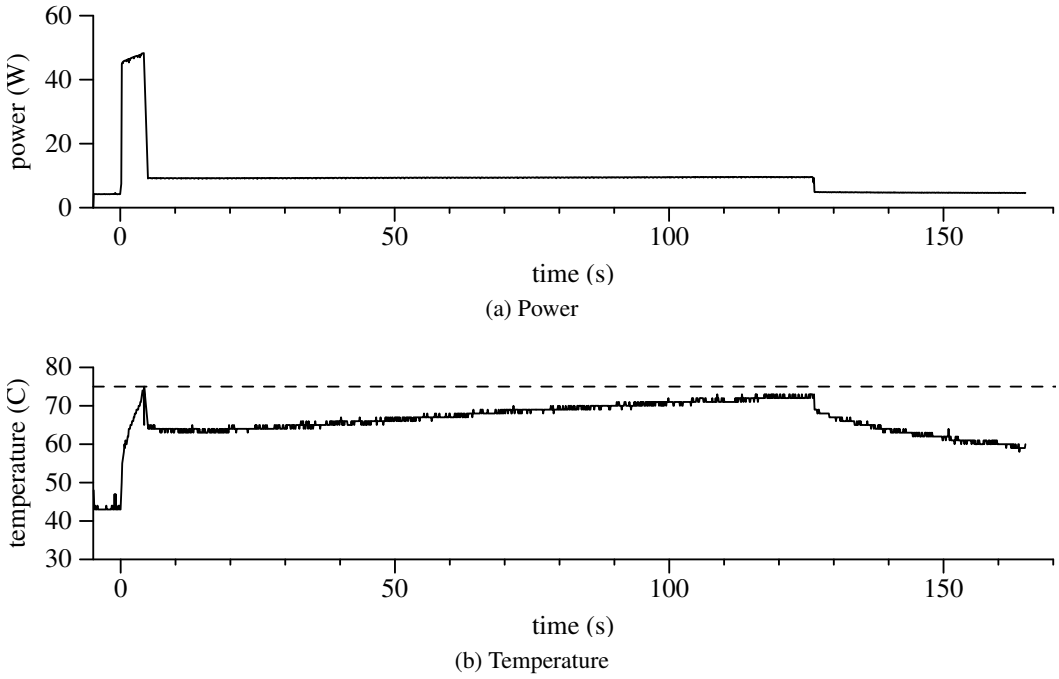
Figure 5.8 compares the estimated and observed sprint durations for all the sprinting configurations available on the testbed. Although the observed and expected sprint times match approximately, there are two regions where they diverge by a small amount: (i) for lower intensity sprints (left of the graph), the thermal capacitance from the heat-spreader underestimates the actual sprint duration, and (ii) for high intensity sprints (right of the graph), the available thermal capacitance from the heat-spreader overestimates the sprint duration.

This divergence in observed and estimated sprint durations reflects on the total thermal capacitance available to each sprint. For the higher intensity sprint, 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 [108]—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 Figure 5.6a. In contrast, when the rate of heating is more gradual, the heat spreads to fully utilize not only the copper heat spreader, but also more distant sources of thermal mass such as the circuit board.

Thus, the total energy expendable during a sprint—and hence the total computation that can be performed while sprinting—can vary depending on the intensity as well and energy efficiency of sprinting. The above result motivates the gradual sprint pacing policy described in Section 6.5.

5.4 Truncating Sprints when the Chip Reaches Threshold Temperature

The discussion thus far only considered the desirable case of *unabridged sprints*: when the system completes all computation while sprinting and starts to idle before the chip gets too hot. However, if a computation lasts long enough for the chip to approach temperature thresholds, the testbed runtime must *truncate* the sprint by dropping the system to its sustainable baseline configuration.



(a) Power

(b) Temperature

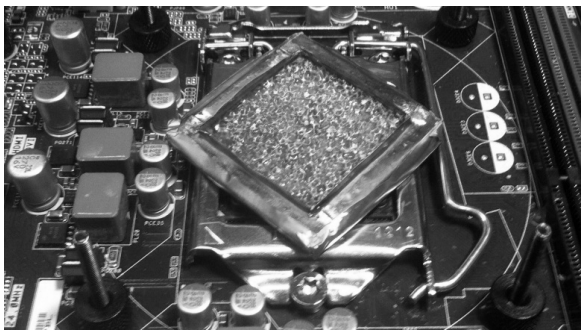
Figure 5.9: Power and thermal response for truncated sprints.

5.4.1 Implementing Sprint Truncation

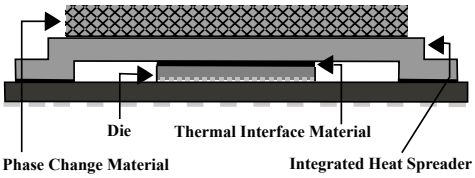
The testbed implements a runtime process to monitor die temperature by querying the on-die temperature sensor every 100 ms. The runtime spawns the workload process, and the two processes can communicate through shared memory (`mmap`-ed into each process's address space). Although implemented at the user-level in this 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 (thereby forcing operating system to migrate all threads to that core), (*ii*) disabling the now-idle cores, and (*iii*) setting the remaining core to its lowest configurable frequency. The testbed software implements these steps using system calls and the standard Linux ACPI interface.

5.4.2 Thermal Response of Truncated Sprinting

Figure 5.9a and Figure 5.9b show the impact of sprint truncation on chip power and temperature over time for both sustained execution and a truncated sprint. As before, temperature rises sharply



(a) Phase change material on top of the package



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

Figure 5.10: Testbed augmented with phase-change material.

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 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 idles, and the chip begins to cool.

5.5 Extending Sprint Duration with Phase-change Material

One approach to delaying, or even avoiding sprint truncation is to augment the system with additional thermal capacitance. As seen in Chapter 4, the latent heat of phase change can offer potentially large heat buffering with small quantities of material; 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 [181].

To test the potential of phase change materials to extend sprint duration, this section describes proof-of-concept experiments using the testbed. Due to the unavailability 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) the following experiments test

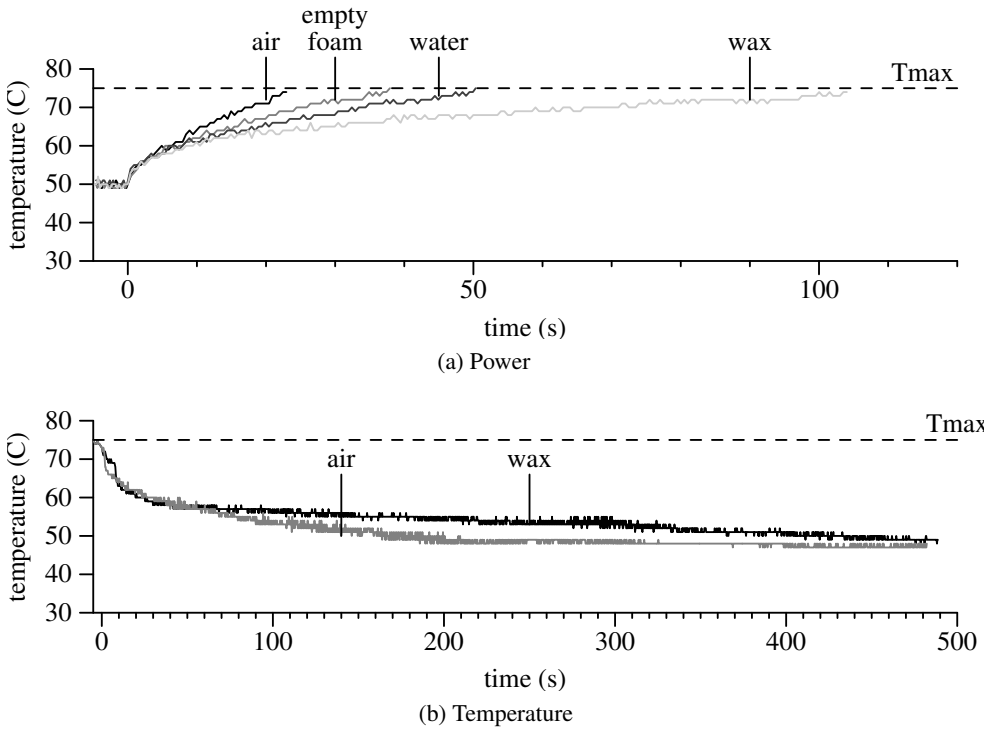


Figure 5.11: Heating and cooling transients for augmented testbed.

---

paraffin (BW-40701 from BlendedWaxes, Inc.) which has a melting point of 54° C. Because paraffin has poor thermal conductivity (0.2 W/mK), it is infused in 0.9 g of Doucel aluminum foam (bulk thermal conductivity of 5 W/mK). To prevent leakage, the paraffin/foam structure is enclosed in a 4.2 cm × 4.2 cm × 0.3 cm box of 0.013 cm thick copper weighing 4 g. Mounting this enclosure on the processor socket using screws provides firm attachment and improves interfacial heat transfer. Figure 5.10 illustrates the above setup.

5.5.1 Heating Transient

Figure 5.11a shows the thermal response of sprinting with 20 W 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, Figure 5.11a also shows the effect of replacing the PCM with an equal weight of water (and a plastic cap to prevent the water from evaporating). The sprint duration with water is 50 s. As the specific heat of water (4.2 J/gK) is higher than that of paraffin (2 J/gK), the most likely conclusion is that the latent heat of the PCM must account for the substantial sprint extension.

5.5.2 Cooling Transient

Figure 5.11b shows the corresponding cool down trends for both the baseline and PCM-augmented testbed. As seen earlier, the baseline converges to the initial temperature ($50°C$) in approximately 200 s. In contrast, the wax-augmented system cools down more slowly. The flattening of slope in temperature is indicative of the freezing phase of the PCM. In all, it takes approximately 10 minutes for the temperature to return to the initial value of $50°C$.

Experiments with maximum sprint intensity (50 W) found the PCM to be less effective in extending sprints limited by heat transfer into the PCM. Hence, although the above 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.

5.6 Limitations of Testbed

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 [34]. 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 [27, 47, 54, 162, 169], a future mobile chip with more cores than can sustainably operate within the thermal constraints of a mobile device seems entirely plausible.

The testbed also completely isolates the electrical considerations to sprinting because the supply, motherboard, and processor are all capable of easily supplying 50 W. The voltage regulator and on-chip power rails are also designed to reliably switch between the various operating configurations. Because it is fabricated using the 32nm technology node, the processor also does not reflect the reliability characteristics of future low-power CMOS technology nodes. In summary, the processor used in the testbed is designed to sustainably operate at high power; its scope in this dissertation is hence limited to the thermal constraints explored in this chapter, and the hardware/software studies described in the next chapter.

5.7 Chapter Summary

This chapter explored the thermal response when sprinting using an experimental testbed constructed by modifying a real system. By constraining heat dissipation to only the lowest operating point of a desktop processor, the testbed creates a thermal and hardware/software environment that can sprint by up to 5× the sustainable power (and 8× peak performance) for a few seconds. Although imperfect in capturing all the constraints of a future mobile device, the large gap in peak-to-sustainable power and performance potentially creates a full-system environment to further investigate sprinting.

- The testbed provides experimental confirmation on a real system that thermal capacitance, in the form of both specific heat of metals, and latent heat of phase-change, can be used to exceed TDP for brief, intense sprints.
- The amount of thermal capacitance available for sprinting could vary depending on sprint intensity because of the time available for heat to spread.
- The total energy expended during a sprint similarly depends on both sprint duration and sprint intensity; the total amount of computation that can be performed during a sprint is therefore sensitive to the relative energy-efficiency of the mode of sprinting.

Having demonstrated the ability to sprint, the next chapter utilizes the testbed as a platform to evaluate sprinting performance and energy.

Chapter 6

Responsiveness and Energy Consumption of Sprinting on a Real System

Because the testbed allows for both parallelism and voltage-frequency scaling, this chapter explores the performance and energy of both forms of sprinting (`Parallel` and `Parallel+DVFS`). Further, sprints are classified into those that complete the entire computation (*unabridged* sprints), and those that need to be *truncated* because the computation is too long to fit within the duration of a single sprint. This chapter is structured as follows:

- Section 6.2 considers the straightforward case of unabridged sprints by sizing workload inputs to be small enough to complete within the expected sprint duration. The findings confirm the responsiveness improvements seen in the earlier simulation-based evaluations. On the testbed system, both `Parallel` and `Parallel+DVFS` sprinting consume less energy to perform the computation than a baseline which does not sprint; although including the post-sprint idle energy causes `Parallel+DVFS` sprinting to be energy inefficient, `Parallel` sprinting still results in average energy savings over the non-sprinting baseline (6% on average) by "sprinting-to-rest."

- Finishing computation early allows the system to idle sooner, saving on "background" energy from components like a shared cache and interconnect that otherwise remain powered-on for

longer durations. To explain the energy-savings potential of sprinting, Section 6.3 describes a simple model capturing the relationship between the speedup achievable when sprinting, and compute, background and idle power.

- Section 6.4 then turns attention to the case of truncated sprints, focusing on two aspects of sprint truncation. First, experiments show that the naive mechanism of truncating parallel sprints by migrating still-active threads to a single core can cause substantial slowdowns. This chapter proposes a sprint-aware, work-stealing runtime as a general strategy to eliminate this *oversubscription* penalty, thereby enabling sprinting to strictly "do-no-harm."

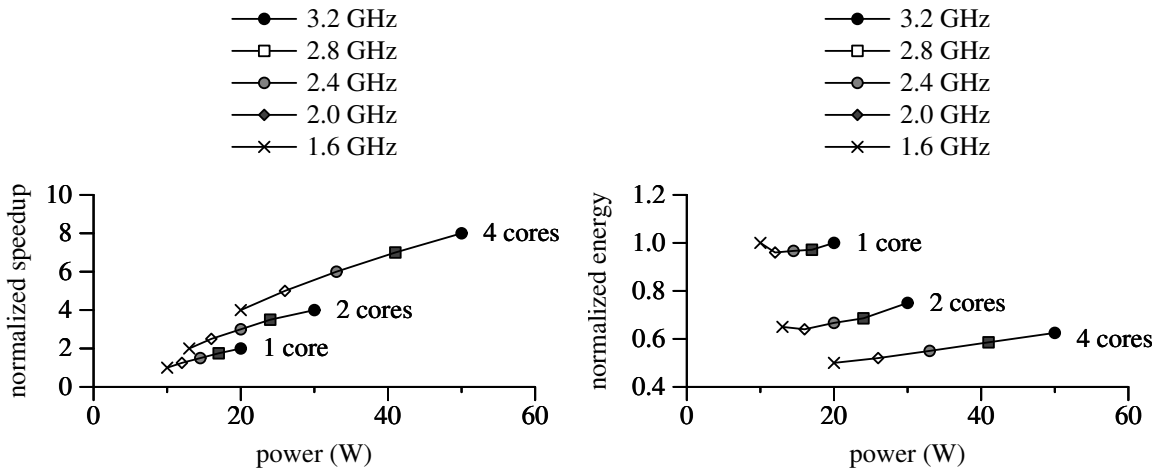
- Although addressing the oversubscription problem ensures that sprinting will not result in performance worse than sustainable single-core execution, simply sprinting at maximum intensity is not always best. Rather, experiments show that maximum responsiveness requires *sprint pacing* in which a sprint 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. Section 6.5 presents two example implementations to demonstrate sprint pacing.

- Based on the energy-savings potential of sprinting, this chapter motivates an alternate computation regime even for sustained workloads: repeatedly sprinting and resting with a sustainable *average* power outperforms conventional operation with constant, sustainable power in thermally limited contexts like the testbed. Section 6.6 analyzes and experimentally evaluates such a sprint-and-rest mode of operation.

To set the context for the subsequent experiments, Section 6.1 first characterizes the testbed's power, peak-performance and energy-efficiency for varying core-frequency settings.

6.1 Testbed Characterization

This section provides the context to interpret the evaluations later in this chapter by first characterizing the expected energy and peak-performance of the testbed at varying operating modes in Sec-



(a) Peak-speedup vs. power

(b) Relative energy vs. power

Figure 6.1: Power, performance, and energy characteristics of testbed.

tion 6.1.1. These characteristics subsequently motivate an analytical model which explains when sprinting can save energy (Section 6.3).

6.1.1 Estimating Peak Performance and Energy

Section 5.1.2 briefly overviewed the range of computing power across various core-frequency configurations of the testbed processor. It is further straightforward to estimate key metrics such as speedup and energy. The relative peak performance (speedup) of two configurations can be estimated as the product of the ratio of number of cores in each configuration and the ratio of the operating frequencies of each configuration. (The estimate is only representative of workloads which are largely parallel and compute bound, such as the evaluation workloads used in this dissertation). Similarly, the relative energy for performing a computation using two different configurations can be calculated given the active power (Figure 5.2) and the speedup (estimated as above).

For example, when computing with four cores at a frequency of 3.2 GHz, the expected peak-speedup over the non-sprinting (one core, 1.6 GHz) baseline is 8× (4× ratio in cores, and 2× in frequency). If the baseline completes a certain computation in ten seconds, then the total energy consumed is 100 J (10 W x 10 s). In contrast, the higher power configuration expends 50 W while computing, but is expected to complete the computation eight times faster, *i.e.*, in 1.25 s, for a

total energy of 62.5 J; hence the relative energy (computed as the ratio) is 0.625×. Figure 6.1a and Figure 6.1b illustrate the peak-performance and energy for each of the active power modes normalized to the TDP constrained baseline.

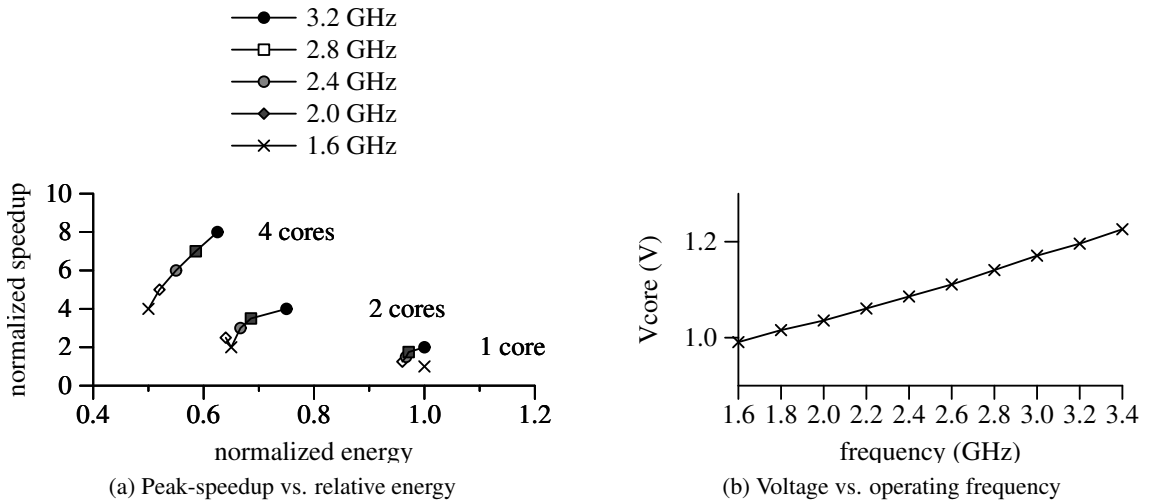
In both these plots, the x-axis indicates growing power (from left to right). On the thermally constrained testbed, only the lowest power (leftmost point at 10 W) is sustainable, and all other configurations can hence only be utilized for temporary operation when sprinting.

6.1.2 Power-constrained Performance and Energy

Because the higher power configurations employ additional cores and/or frequency, each factor contributes linearly to expected speedup, and hence the normalized speedup of any non-baseline configuration exceeds 1× (Figure 6.1a). The pareto-optimal frontier indicates the peak performance achievable by a TDP constrained configuration. For example, under the testbed conditions of 10 W heat dissipation, the maximum speedup achievable is 1× because only the single core, 1.6 GHz baseline is sustainable; as the operating power increases, the highest performing modes move to two cores at 1.6 GHz (at ≈13 W), and two cores at 2.0 GHz (at ≈16 W). Further increase in power results in maximum speedup when computing with all four cores.

Relative energy of activating multiple cores. Surprisingly, despite operating with higher power, the non-baseline configurations consume *less* energy than the baseline to perform the same computation (Figure 6.1b). The relative energy across different core counts for the same frequency (1.6 GHz) decreases from 1× for a single core (normalized baseline) to 0.6× with two cores and 0.5× for four cores; *i.e.*, quadrupling the number of cores at minimum frequency results in a doubling in energy *savings* with ideal speedup. This trend is consistent with previously published estimates [13] of sub-linear increase in power as core counts grow: whereas a single core at 1.6 GHz draws ten watts, it requires approximately only three additional watts to activate each additional core (13.5 W package power for two cores, and 20 W for four cores). Hence, computing with four cores at baseline frequency provides improves computation efficiency by drawing only twice as much power, yet enabling four times the performance as the sustained baseline.

Relative energy of voltage-frequency boosting. Figure 6.1b also shows the relative energy of computing with varying frequency. For two and four core configurations, the lowest frequency consumes the least energy, with energy increasing non-linearly with increasing frequency. For the



(a) Peak-speedup vs. relative energy

(b) Voltage vs. operating frequency

Figure 6.2: Relative energy-performance of different modes of operation, and voltage-frequency relationship on testbed.

single core configuration, the energy at the intermediate frequencies (all frequencies excepting the minimum and maximum) is moderately less than the baseline; for example the expected peak-performance at 2 GHz (2/1.6 = 1.25×) exceeds the corresponding increase in power (12 W/10 W = 1.2×) resulting in 4% energy savings. However, the doubling in frequency (1.6 GHz to 3.2 GHz) is accompanied by a corresponding doubling in power (from 10 W to 20 W) so that the relative energy remains 1×.

To summarize, for a single core, energy consumption varies by less than 4% across a 2× range in frequency; for two and four cores, the lowest frequency is clearly the most energy efficient operating point. Overall, the results largely conform to expected trends from Chapter 2 that frequency boosting is less energy-efficient than nominal operation. However, the counterintuitive result is that operating power increases by only 2.5× for a doubling in frequency. The reason is that frequency boosting is accompanied by only a small increase in supply voltage. Figure 6.2b shows the core voltage registered for each frequency setting. When doubling frequency from 1.6 GHz to 3.2 GHz, the supply voltage is boosted to only 1.2×. Whereas a naive estimate of active power ($V^2 \cdot f$) predicts a corresponding factor of 2.9× increase in power, the observed increase is only 2.5×.

Figure 6.2a captures the energy-performance characteristics of the testbed. The TDP constrained configuration (the one core, 1.6 GHz point at the bottom right) is neither the most respon-

sive, nor the most energy efficient computing mode. However, sprinting can exceed TDP to expose the higher performance and more energy efficient modes rather than sustained computation under TDP constraints. Although the above inferences are reflective of the concrete power-performance characteristics of the testbed system, similar conditions may prevail in future thermally constrained devices. The testbed baseline configuration is clearly sub-optimal in terms of both energy and performance because the Sandy Bridge processor used in the setup is designed to sustain maximum performance. The "uncore" components such as caches and interconnect are sized to support four cores clocked at 3.2 GHz, and remain active even when only one core is active. Traditional guidelines of scaling the uncore with the number of cores suggest that background power during computation would continue contribute to a significant fraction of total power.

The testbed thus represents a concrete system where higher responsiveness and higher energy-efficiency operation is precluded by thermal design power under conventional execution. The remainder of this chapter evaluates the benefits of utilizing these higher-power configurations to sprint.

6.2 Speedup and Relative Energy of Unabridged Sprints

The first set of experimental evaluations consider the straightforward case when sprints are *unabridged*, that is, when the parallel work can be completed entirely during a sprint without exhausting the system's thermal capacitance. This section explores the responsiveness and energy efficiency of unabridged sprints; Section 6.4 turns to the more complex case of truncated sprints.

6.2.1 Experimental Methodology

The evaluation uses the workloads from the simulation studies in Chapter 4, additionally introducing a speech recognition workload (`speech`) which is based on ALPBench SpeechRec benchmark[103] (itself derived from the CMU Sphinx project [139]).

Unlike a software simulator, the testbed is a real-world environment subject to external sources of experimental noise such as physical conditions (*e.g.*, system/ambient temperature, air flow) and intervention from both software (*e.g.*, operating system scheduling, background process activity) and hardware (*e.g.*, thread scheduling, dynamic performance scaling). Precautionary measures mitigate these effects towards setting up an environment for repeatable results with a high degree of

confidence. Following the methodology from the previous chapter, before each experiment, the testbed is initially idle at a temperature of 50°C and all experiments are performed under similar conditions of room temperature and fan speed. The thermal limit for sprinting is set to 75°C; this temperature was chosen to be below the chip manufacturer's ratings ("hot" temperature of 78°C, and "critical" temperature of 98°C) to preclude throttling due to hardware and operating system thermal-trip mechanisms.

To eliminate variable performance due to hardware/operating system interference, the sprinting runtime controls all power mode (P-state/C-state) transitions. Both TurboBoost and hyper-threading are disabled, and workload threads are pinned to dedicated cores to prevent migration; the evaluated workloads spawn at most as many threads as the number of available cores, barring experiments which explicitly address thread multiplexing. The Linux kernel is configured in the tickless mode to minimize wakeups and allow the system to idle. During execution, a runtime process periodically monitors temperature and energy. The monitor process is scheduled to run at 100 ms intervals; whereas monitoring with much larger frequency (10 ms) showed noticeable spikes in execution power traces, the relatively smooth power plots in Figure 5.4a, Figure 5.7a (and subsequent figures in this chapter) confirm that monitoring every 100 ms causes negligible perturbation. Finally, reported runtime and energy measurements are averages from multiple trials. After discarding anomalous executions due to occasional machine perturbation, the results were repeatable within negligible error margins.

The above methodology applies to all experiments in this chapter. For this section (unabridged sprinting), workload inputs are sized so that all computation completes before reaching the thermal limit.

6.2.2 Responsiveness Benefits of Sprinting with Maximum Intensity

The testbed, like any computing platform, is expected to be most responsive when all computation is executed using the highest performance mode. Figure 6.3a 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 6.1× benefit over the baseline. The performance results are similar to the simulated responsiveness benefits from Chapter 4; with the workloads being largely parallel and compute intensive, sprinting to completion is able to achieve near

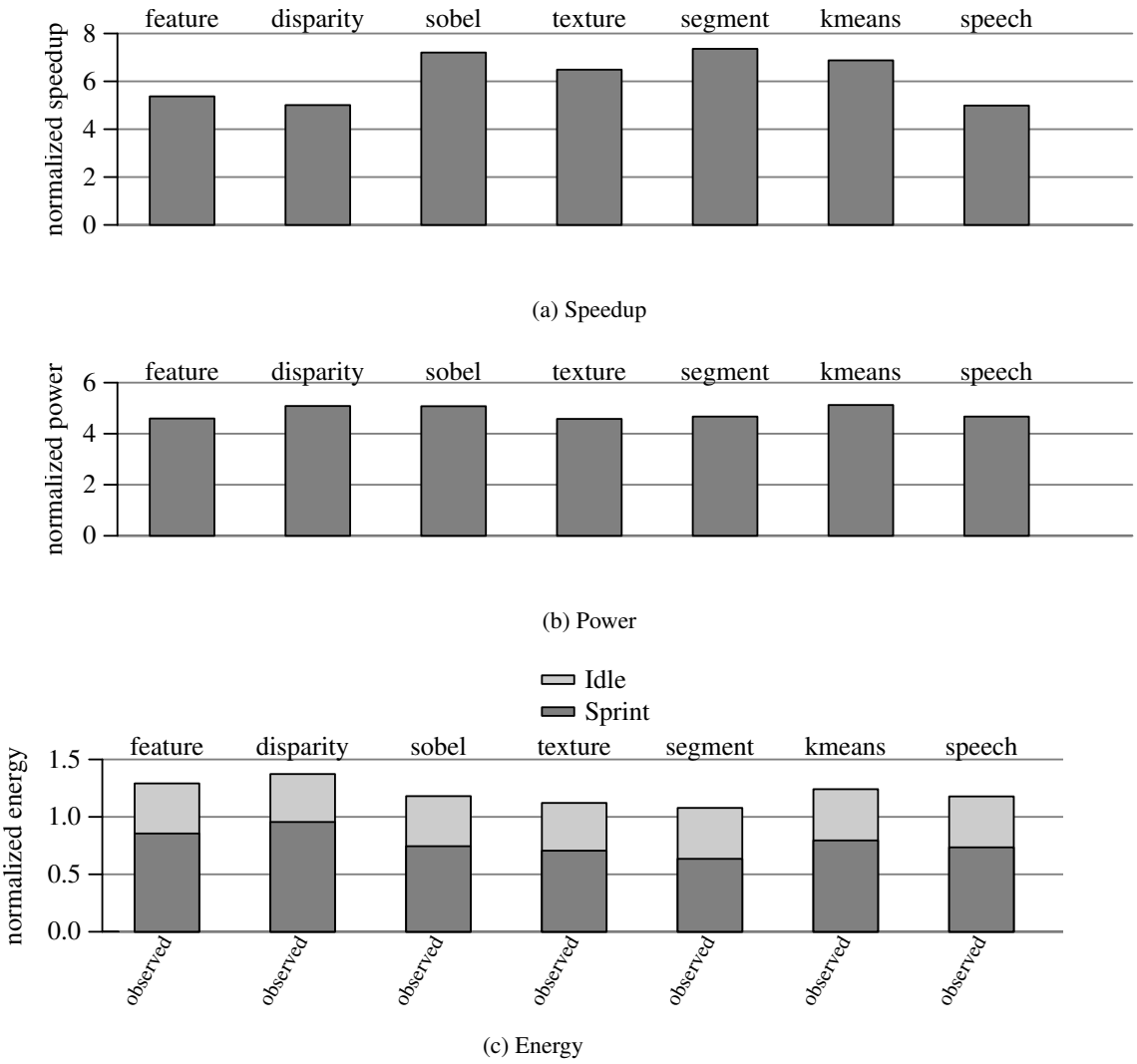


Figure 6.3: Speedup, power, and energy (normalized to the one-core 1.6 GHz sustainable baseline) for four cores at 3.2 GHz.

---

peak-performance (*i.e.*, close to the back-of-the-envelope 8× peak-speedup estimate). 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.

6.2.3 Energy Impact of Sprinting with Maximum Intensity

Because the energy expended during a sprint depends on sprint power, Figure 6.3b first shows the sprint power measured during execution of each workload with the most intense sprint (normalized

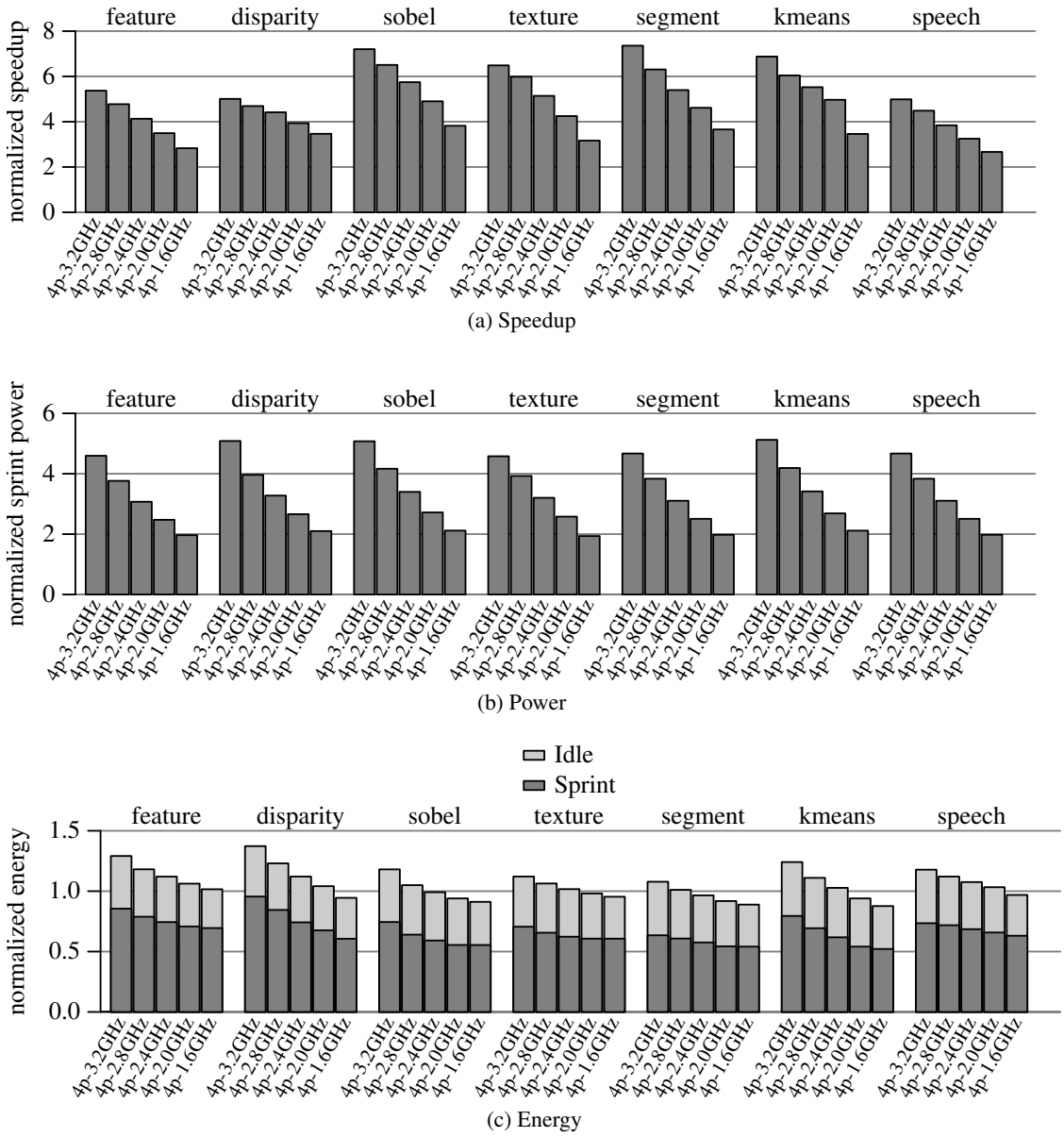


Figure 6.4: Speedup, power, and energy (normalized to the one-core 1.6 GHz sustainable baseline) for four cores across frequencies.

---

to the non-sprinting baseline power). The immediate observation is that sprint power is largely invariant (approximately 50 W, similar to Table 6.2) across the workloads. However, because the workloads exhibit dissimilar speedups, the relative energy for computation is also expected to vary across the workloads.

Sprint energy. Ignoring idle energy for the moment, the lower stacked component (dark) in the first set of each pair of bars in Figure 6.3c shows the energy (normalized to the non-sprinting baseline) to complete the computation task of each workload. Overall, the experiment shows that sprinting reduces the energy required to complete computation by up to 30%, with average energy savings of 23% over all workloads when compared to executing the same computations on a non-sprinting baseline. The reason for this surprising result is that a significant fraction of total energy goes towards powering on background components. On this testbed system the background power comprises uncore components like the shared 8MB cache and interconnect which remain powered on regardless of the core computational resource; on mobile systems this background power may comprise non-compute resources like the screen and radio which remain powered on as long as the user is engaged. By increasing compute power, sprinting better amortizes the effective contribution of the constant background power to total energy.

Implications of idle energy. Only comparing the active computation power of the sprinting and sustainable modes is unfair in that the energy is not measured over an equal time interval. After completing the computation within a single unabridged sprint, the testbed system returns to its idle state. Unfortunately, chips continue to dissipate power even when idle. By finishing earlier, the chip must idle for longer [118]; Thus, to facilitate a fair energy-efficiency comparison, the total energy of the sprinting system should also account for idle power for the remaining duration when the sustainable baseline is still active. The upper components of each bar in Figure 6.3c shows the idle energy over an equalized time window. Considering total energy, maximum intensity sprinting is now less energy-efficient than the sustainable baseline for every workload, and 21% lower on average, implying that the per-op energy efficiency from increased compute power while sprinting is insufficient to compensate for the additional energy required for voltage-frequency scaling.

6.2.4 Unabridged Sprints with Lower Frequency

Incorporating all cores at maximum frequency/voltage is energy inefficient because of the causes a super-linear power increase with voltage/frequency scaling. Sprinting with lower frequencies is thus an alternative approach to seek improvements in energy efficiency. Figure 6.4 shows the responsiveness, power, and energy implications of sprinting with all four cores at various frequency/voltage levels. (Based on Figure 6.2a, the four core configurations dominate lower core counts,

hence non-sustainable configurations involving fewer cores are omitted.) The left-most bar in each group of Figure 6.4a shows the same speedup (6.1× on average) with a maximum-intensity four-core 3.2 GHz sprint that was presented earlier in Figure 6.4. The corresponding bars in Figure 6.4b show the 5× power increase for such a sprint. As the frequency is lowered, the responsiveness improvements decrease, but power decreases relatively more.

Figure 6.4c shows the energy implications of each sprint, apportioned into active (sprinting) and idle 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. Considering active energy alone, the potential energy savings with lower frequency sprints grows to nearly 40% (compared to 23% for the maximum intensity sprint). This comparison does not include idle energy, implying that the *energy-per-operation* is lower at lower frequencies. Even including idle energy, the lowest-frequency four-core sprint (1.6 GHz) results in an efficiency gain of 6%. Hence, when sprint intensity is selected appropriately, sprinting can improve energy efficiency as well as responsiveness even on today's chips. The key insight here is similar to previously observed savings attributed to making systems "race-to-idle" [12, 14, 55, 57, 106, 112, 118]. Concretely, when idle power is low enough, speeding up computation to save background energy by powering off uncore components sooner enables energy savings. In thermally constrained environments like the future projected for mobile chips, sprinting can utilize dark silicon to save energy in addition to enhancing responsiveness. Section 6.3 specifies the conditions for sprinting using an analytical model.

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 [124] and ARM's big.LITTLE multicores [61]), there exists a substantial potential for sprinting as an energy saver as well as a responsiveness enabler.

6.3 When Does Sprinting Save Energy?

To further understand the energy-efficiency of sprinting, this section derives a simple model relating energy with speedup, active power, background power, and idle power. The opportunity to save

Parameter	**D**erivation	**M**eaning
N	Input	Number of cores
f	Input	operating frequency
f_{min}	Input	Minimum operating frequency
$t_{compute}(N,\ f)$	Input	Computation time with N cores at frequency f
P_{idle}	Input	Idle power
$P_{uncore}(f)$	Input	Background power at frequency f
$P_{core}(1,\ f)$	Input	Core power with 1 core at frequency f
$P_{core}(N,\ f)$	$N \cdot P_{core}(1,\ f)$	core power with N cores at frequency f
$P_{total}(N,\ f)$	$P_{core}(N,\ f) + P_{uncore}(f)$	Sprint (total) power with N cores at frequency f
$P_{sustainable}$	Assumed as $P_{total}(1,\ f_{min})$	Maximum thermally sustainable power
$S(N,\ f)$	$\frac{t_{compute}(N,\ f)}{t_{compute}(1,\ f_{min})}$	Speedup at N cores and frequency f relative to baseline at 1 core and f_{min}
$t_{idle}(N,\ f)$	$t_{compute}(N,\ f) - \frac{t_{compute}(N,\ f)}{S(N,\ f)}$	Idle time after computing with N cores at frequency f
$E_{compute}(N,\ f)$	$P_{total}(N,\ f) \cdot t_{compute}(N,\ f)$	Energy required for active computation with N cores at frequency f
$E_{idle}(N,\ f)$	$P_{idle} \cdot t_{idle}(N,\ f)$	Energy spent idling after computing with N cores at frequency f
$E_{total}(N,\ f)$	$E_{compute}(N,\ f) + E_{idle}(N,\ f)$	Total energy across time required for computation with N cores at frequency f
$r_{sprint}(N,\ f)$	Upper-bound: $\frac{P_{sustainable} - P_{idle}}{P_{total}(N,\ f) - P_{idle}}$	Fraction of total time spent in sprint mode

Table 6.1: Parameters used in energy analysis.

energy with sprinting arises because of the power required to keep shared (i.e., uncore) components of a chip active to support the operation of even a single core. Despite being classified as an overhead, this background power contributes to acceptable performance. For instance, last level caches and interconnects reduce miss rate and penalty by staging and moving data to the core. In the system used for evaluations in this chapter, uncore power is twice the core power at minimum operating frequency (Table 6.2). System designs embrace this overhead and seek to amortize it by scaling up computation resources (*e.g.*, adding more cores) to compute more energy-efficiently per operation—a similar argument was made for the *cost* of parallel computing systems by Wood and Hill [177]. Because of this background power, speeding up computation can save energy by "racing-to-idle" and reducing the time for which the background components remain active [12, 106, 112, 118].

Although it is desirable to operate in the most energy efficient modes, the thermal constraints that give rise to dark silicon can also preclude such sustained operation. By sporadically activating dark silicon, computational sprinting can reduce the energy-per-operation and "sprint-to-idle". This section presents an analytical model using the parameters in Table 6.1. The model separates core

Frequency	Cores	Active Power (W)	Relative Power	Core Power	Uncore Power	Std. Error
1.6 GHz	1	10 W	1.0x	3.4 W	6.6 W	0.02 W
	2	13.3 W	1.33x			
	4	20.1 W	2.01x			
2.0 GHz	1	11.5 W	1.15x	4.5 W	6.9 W	0.05 W
	2	15.9 W	1.59x			
	4	25.1 W	2.51x			
2.4 GHz	1	13.9 W	1.39x	6.6 W	7.3 W	0.06 W
	2	20.7 W	2.07x			
	4	33.8 W	3.38x			
2.8 GHz	1	16.1 W	1.61x	8.2 W	8.1 W	0.16 W
	2	24.8 W	2.48x			
	4	40.9 W	4.09x			
3.2 GHz	1	20.1 W	2.01x	10.1 W	9.9 W	0.09 W
	2	29.9 W	2.99x			
	4	50.2 W	5.02x			

Table 6.2: Testbed power profile.

(active), uncore (background), and idle power based on empirical data from a real system. The workload-dependent variation in power consumption is not explicitly included in the model, because experiments show low variation across the compute intensive workloads used in the evaluation (Figure 6.4b). For a given frequency, the testbed system closely fits a model with a fixed background power and active power growing linearly in direct proportion to the number of active cores (N) (see Table 6.2). Both background power ($P_{uncore}(f)$) and active core power ($P_{core}(N,\ f)$) vary with frequency. The model compares the energy of sprinting relative to a sustainable baseline execution at minimum power with one core at f_{min} (*i.e.*, at power $P_{core}(1,\ f_{min})$) while obtaining a speedup of $S(N,\ f)$.

This model analyses the energy impact of sprinting, particularly focusing on the questions (i) when does sprinting improve energy efficiency per operation, (ii) when does sprinting result in a net energy savings when also considering the implications of non-negligible idle power.

6.3.1 Sprinting to Reduce Energy-per-Operation

The total energy a system consumes while computing is:

$$Energy\ during\ computation = (core\ power + background\ power) \cdot compute\ time$$

To compare energy relative to the baseline execution with a single core operating at frequency f_{min}, the core power and compute time can be expressed in terms of their baseline counterparts:

$$\begin{aligned} E_{compute}(N,\ f) &= (P_{core}(N,\ f) + P_{uncore}(f)) \cdot t_{compute}(N,\ f) \\ &= (N \cdot P_{core}(1,\ f) + P_{uncore}(f)) \cdot \frac{t_{compute}(1,\ f_{min})}{S(N,\ f)} \end{aligned}$$

By setting N to 1 and f to f_{min}:

$$E_{compute}(1,\ f_{min}) = (P_{core}(1,\ f_{min}) + P_{uncore}(f_{min})) \cdot t_{compute}(1,\ f_{min})$$

Thus, the relative energy is:

$$Relative\ Energy = \frac{E_{compute}(N,\ f)}{E_{compute}(1,\ f_{min})} = \frac{N \cdot P_{core}(1,\ f) + P_{uncore}(f)}{S(N,\ f) \cdot (P_{core}(1,\ f_{min}) + P_{uncore}(f_{min}))} \quad (6.1)$$

To compute more energy efficiently ($Relative\ Energy < 1$), the higher power sprint modes must therefore deliver a minimum speedup:

$$S(N,\ f) > \frac{N \cdot P_{core}(1,\ f) + P_{uncore}(f)}{P_{core}(1,\ f_{min}) + P_{uncore}(f_{min})} \quad (6.2)$$

For the particular case of sprinting by activating additional cores without frequency scaling ($f = f_{min}$), the minimum required speedup can be expressed in terms of the ratio of background power to core power as:

$$S(N,\ f_{min}) > \frac{N + \frac{P_{uncore}(f_{min})}{P_{core}(1,\ f_{min})}}{1 + \frac{P_{uncore}(f_{min})}{P_{core}(1,\ f_{min})}} \quad (6.3)$$

With no background power, ideal, linear speedup ($S(N,\ f_{min}) = N$) is required for any additional cores to even be energy-neutral with single core operation. However, non-zero background power reduces the minimum speedup required—the increase in the denominator is much larger than the corresponding increase in the numerator. Therefore, with higher background power, even sub-linear speedup can be more energy-efficient than operating in the lowest-power mode. For example, in the evaluation system, where $P_{uncore}(f_{min})$ is 6.6 W and $P_{core}(1,\ f_{min})$ is 3.4 W, a speedup exceeding 2× with four cores is sufficient for saving energy.

6.3.2 Implications of Idle Power

The above analysis considered only the energy spent while the system is active. However, after completing a task sooner by sprinting, the system returns to its idle state, which typically incurs non-zero idle power. A more conservative model needs to consider this idle energy and compare total system energy over the same time period as the slower baseline execution.

$$
\begin{aligned}
E_{total}(N,\ f) &= E_{compute}(N,\ f) + E_{idle}(N,\ f) \\
E_{idle}(N,\ f) &= P_{idle} \cdot (t_{compute}(1,\ f_{min}) - \frac{t_{compute}(1,\ f_{min})}{S(N,\ f)}) \\
E_{total}(1,\ f_{min}) &= E_{compute}(1,\ f_{min})
\end{aligned}
$$

Thus, the relative energy is:

$$
Relative\ Energy = \frac{E_{total}(N,\ f)}{E_{total}(1,\ f_{min})} = \frac{N \cdot P_{core}(1,\ f) + P_{uncore}(f) + P_{idle} \cdot (S(N,\ f) - 1)}{S(N,\ f) \cdot (P_{core}(1,\ f_{min}) + P_{uncore}(f_{min}))} \tag{6.4}
$$

The minimum speedup required for core-only sprinting to be energy efficient is therefore:

$$
S(N,\ f_{min}) > \frac{N + \frac{P_{uncore}(f_{min}) - P_{idle}}{P_{core}(1,\ f_{min})}}{1 + \frac{P_{uncore}(f_{min}) - P_{idle}}{P_{core}(1,\ f_{min})}} \tag{6.5}
$$

Equation 6.5 is similar to the previous requirement on speedup (Equation 6.3), except that the background power is now offset by P_{idle}; if the idle power is zero, the two equations are identical. We typically expect idle power to be less than background power in most reasonably engineered systems. In a sprint-enabled system, when sufficient speedup is obtained, it can be possible to utilize dark silicon to "sprint-to-idle" to save energy—the opportunity for saving energy grows with the difference between background and idle power. For example, in the evaluation system, where P_{idle} is 5 W ($P_{uncore}(f_{min})$ and $P_{core}(1,\ f_{min})$ are 6.6 W and 3.4 W as stated previously), speedup exceeding 3× with four cores is sufficient for saving energy.

6.3.3 Comparison with Observed Energy

The estimates from the above equations are directly comparable to the evaluated results, given the speedup per workload. As an example, consider the maximum intensity sprint (four cores, 3.2 GHz) from Section 6.2.

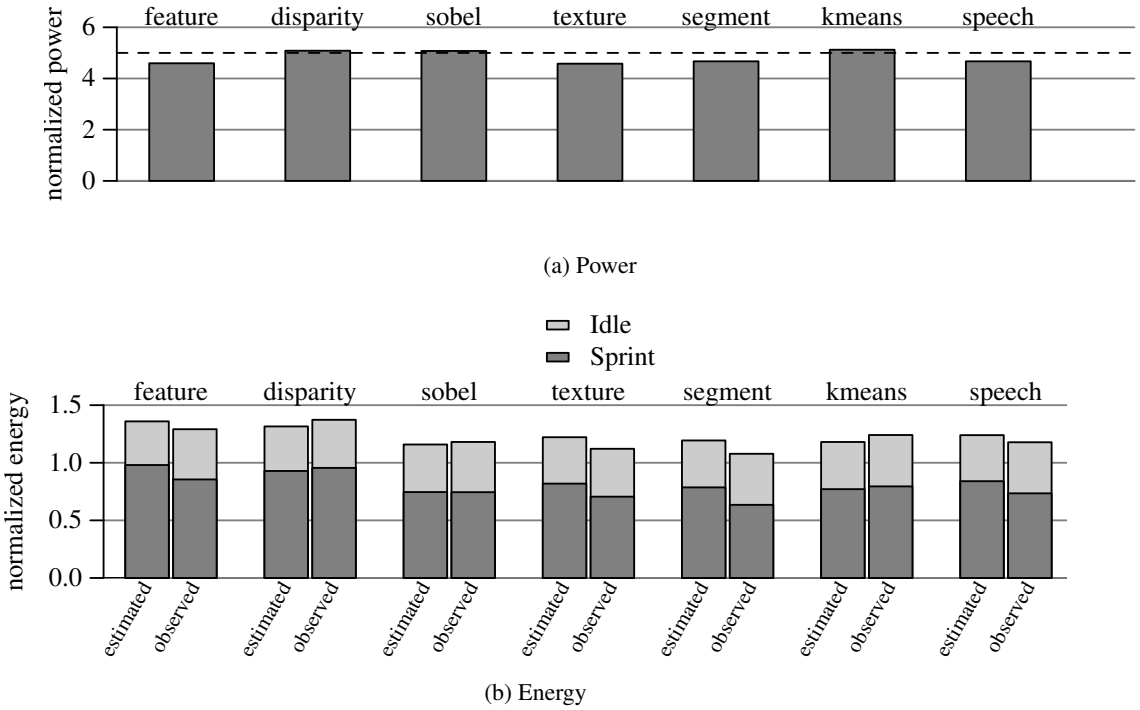


Figure 6.5: Estimated and measured energy (normalized to the one-core 1.6 GHz sustainable baseline) for four cores at 3.2 GHz.

The lower stacked component (dark) in the first set of each pair of bars in Figure 6.5b shows the estimated energy (normalized to the non-sprinting baseline) to complete the computation task of each workload. Because the expected sprint energy lies below the 1.0, sprinting is expected to be more energy efficient than the baseline. The measured energy (bar labeled `observed` in Figure 6.5b) largely confirms these estimates. In `feature`, `texture`, `segment` and `speech`, the model overestimates sprint energy because the observed power is less than the assumed 50 W (below the marker line at 5× normalized power in Figure 6.5a), whereas for `kmeans`, `disparity` and `sobel`, the compute power slightly exceeds 50 W causing the model to underestimate the relative energy. Overall, the experiment confirms that the background energy conserved by completing sooner more than compensates for the super-linear cost of voltage scaling.

The upper component of each bar in Figure 6.5b shows the idle energy over an equalized time window between the sprinting and baseline executions. As seen earlier, maximum intensity sprinting is less energy-efficient than the sustainable baseline for every workload, and 21% lower on

average. In fact, extrapolating from Equation 6.4, to remain even energy neutral with respect to the baseline, sprinting with 50 W must speedup execution 9×; with a peak achievable speedup of 8×, the maximum intensity sprint is inevitably energy inefficient on this testbed.

Unabridged sprinting can therefore improve both performance and energy. However, not all computation tasks are guaranteed to complete within a sprint. The next section describes sprint truncation for computations that outlast available thermal headroom.

6.4 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.

Section 6.4.1 shows 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; Section 6.4.2 attributes the causes to the penalties of *oversubscription*, where performance can degrade when several threads need to be scheduled on just a single core; Section 6.4.3 explores mitigating these effects using a sprinting-aware task-based parallel runtime.

6.4.1 Performance and Energy Penalties of Sprint Truncation.

To evaluate the impact of sprint truncation, the next experiment varies the length of each computation and measures responsiveness across sprint intensities. In Figure 6.6, each group of bars shows the execution time (Figure 6.6a) or energy (Figure 6.6b) 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.

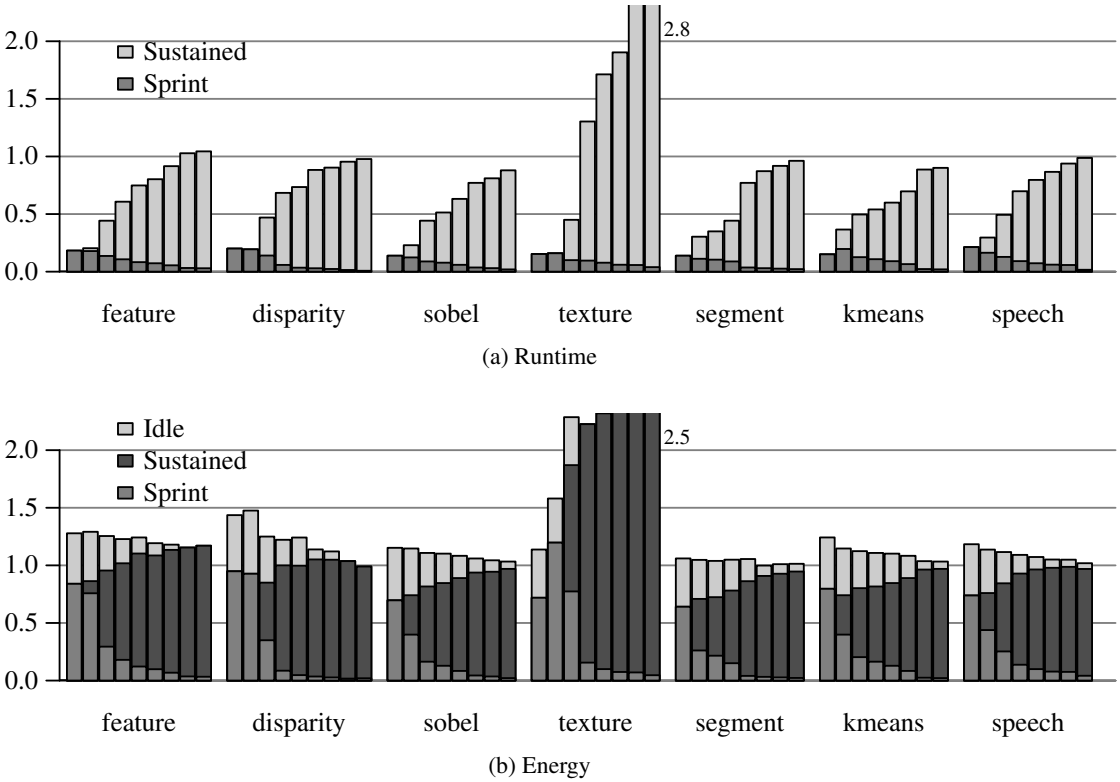


Figure 6.6: Runtime and energy spent during sprinting, sustained, and idle modes for 4-core sprints at 3.2 Ghz (normalized to the one-core 1.6 Ghz baseline.) Bars represent increasing computation lengths from left to right.

6.4.2 Inefficiency 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 6.6a). 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 should "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 [23, 63, 78, 92, 166].

Demonstrating the penalty of oversubscription. To demonstrate the performance penalty of oversubscription, Figure 6.7 shows the performance impact of spawning N threads but pinning

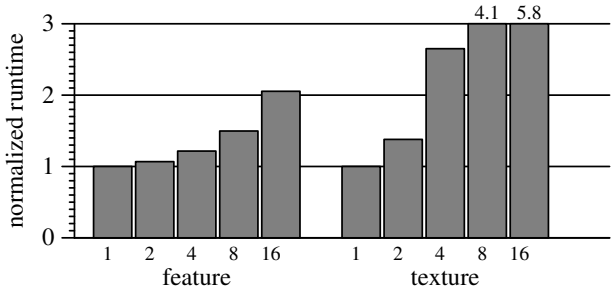


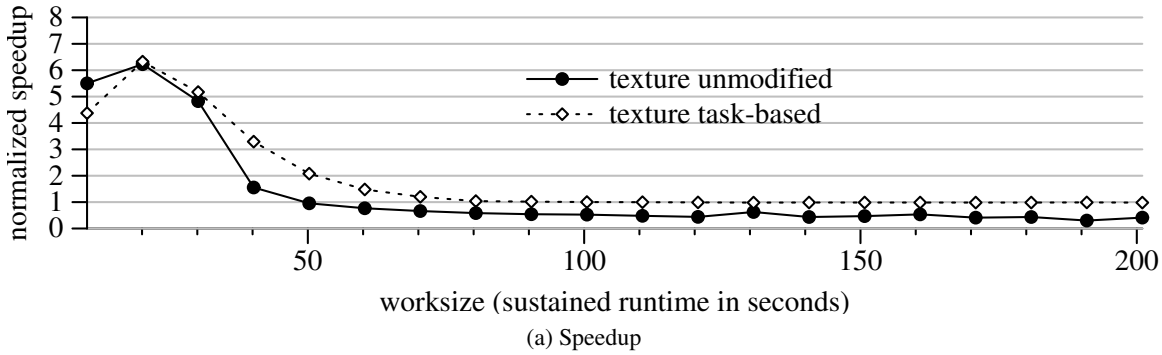
Figure 6.7: Runtime penalty from oversubscription with increasing numbers of threads on a single core.

---

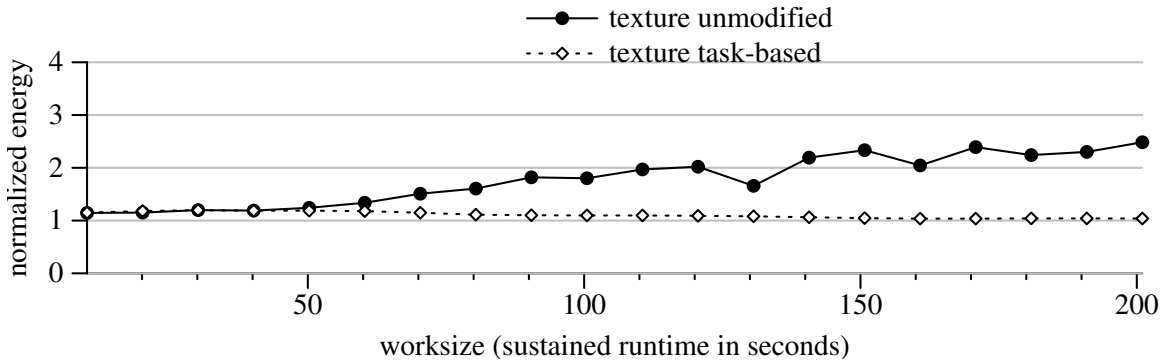
them to a single core. As the amount of oversubscription increases, so does the penalty. Although most of the workloads are not sensitive to oversubscription, the effect is particularly pronounced in `texture`, where the penalty is over 2.4× for a four-to-one oversubscription ratio. The penalty is as high as 5.8× for 16 threads.

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. The reported results already include user-level synchronization primitives that `yield()` after spinning for 1000 cycles. The precise value of the back-off latency is unimportant; although yielding immediately or not yielding at all causes performance and energy overheads, background experiments showed that these metrics were largely insensitive to a wide range of values in between.

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 [166], which is the approach adapted here.



(a) Speedup



(b) Energy

Figure 6.8: Speedup and energy comparison of the unmodified threaded and task-based implementations of `texture`.

6.4.3 Sprint-aware Task-based Parallel Runtime to Mitigate Oversubscription.

Efficient sprint truncation requires an efficient mechanism to dynamically change the number of active software threads. One such is the task-queue based worker thread execution frameworks [3, 24, 39, 53, 98, 166], 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.

The application mitigates oversubscription penalties with a sprint-aware task-stealing runtime framework (similar to Intel's Threading Building Blocks [3]). 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 runtime monitor process, which 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 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, consider a variant of the `texture` workload rewritten for this task-based parallelism model. Figure 6.8 shows the sprint truncation behavior of the original multi-threaded version of `texture` and the task-based variant for various computation lengths. When the computation is shorter than 1 s of baseline execution, task-creation and stealing overheads result in a performance penalty—whereas the unmodified workload is sped up by 5.5×, the task-based workload achieves a speedup of 4.2×. These overheads are amortized for relative longer computations (albeit still short enough that the sprint is unabridged), both versions have similar responsiveness. However, when sprints are truncated for further increasing computation length, the performance of the original multi-threaded version of `texture` falls well below that of the sustainable baseline, whereas 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.

6.5 Sprint Pacing

Section 6.2 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 as simple. Figure 6.9a and Figure 6.9b shows responsiveness benefits and relative energy 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

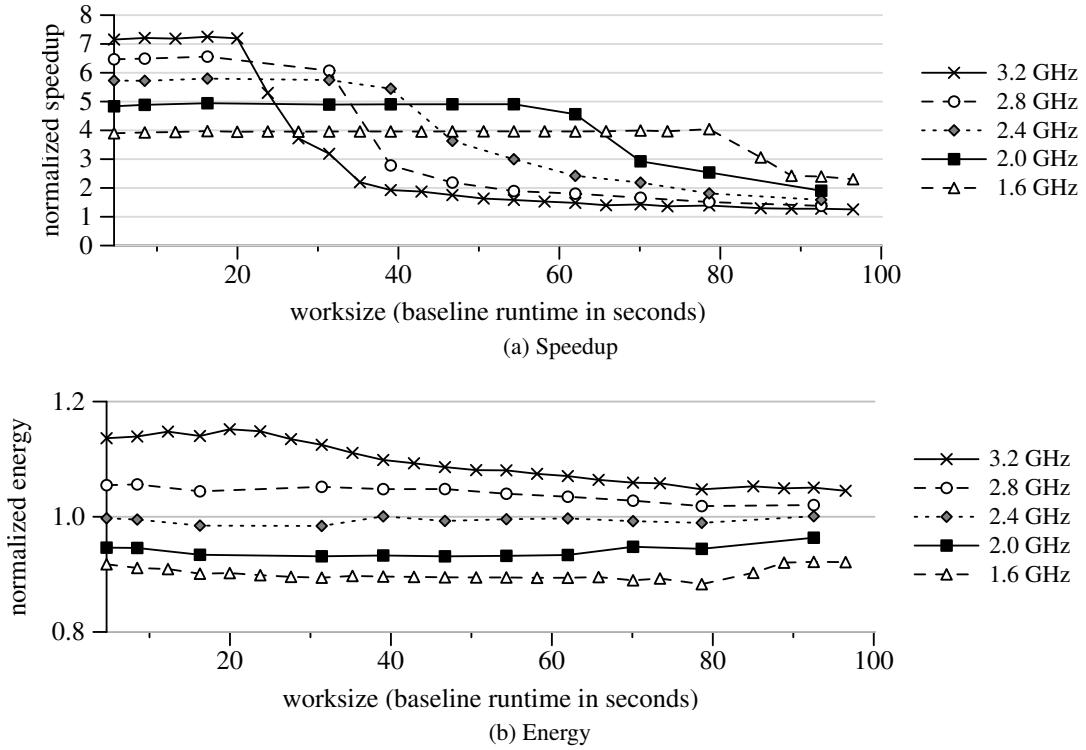


Figure 6.9: Speedup and energy versus size of computation for sprinting with four cores at different differences.

---

execution. However, for intermediate computation lengths, the optimal sprinting mode is not always maximum sprint intensity because the total computation performed during a sprint—the area under the curves—grows as frequency decreases. This section explains the reasons behind this seeming discrepancy, motivates the need for a *sprint pacing policy* which can adapt to computation length, and evaluates example policies for pacing a *single* sprint; Section 6.6 addresses the case of repeated sprint-rest operation.

6.5.1 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 versus 1.6 GHz from Figure 5.8. 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 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 that 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 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.

The most critical impact on sprint pacing policy is the length of the computation, which can guide 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.*, [73]) 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. As seen in Section 3.4, the system can dynamically adapt to execution length based on the thermal state determined by either (i) energy expended, or (ii) temperature of the system. The following experiments illustrate example policies using each of the above approaches (*i.e.*, energy and temperature) to pace sprints.

6.5.2 A Simple, Two-intensity Sprint Pacing Policy

A simplified observation from Figure 6.10 is that short computations (less than 22 s of baseline computation) achieve maximum responsiveness benefits (7.3×) when sprinting most intensely (with all four cores at 3.2 GHz), whereas the longest computations that still achieve speedup do so when sprinting with maximum energy efficiency (sprinting with all four cores at 3.2 GHz results in 4× speedup for up to 80 s of baseline computation). Therefore, one approach to enable maximum responsiveness for short computations and yet achieve significant (4×) speedup for longer computations, is to partially utilize the thermal headroom across either mode of sprinting. A simple energy-based approach is to apportion the thermal capacitance (188 J) so that half the energy (90 J)

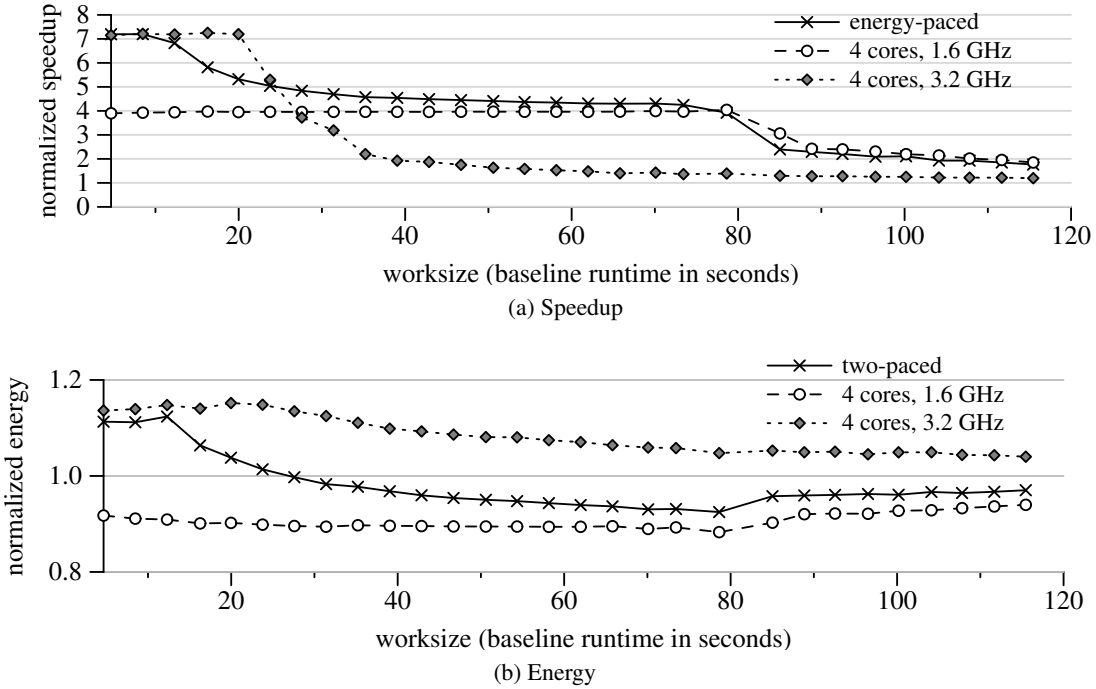


Figure 6.10: Speedup and energy for sprinting based on an allocation of sprint budget between the most responsive and most energy efficient schemes.

is used for maximum intensity sprinting, after which the frequency is reduced to 1.6 GHz. As before, the sprinting runtime monitor throttles execution to the sustainable baseline by deactivating the three additional cores when the thermal headroom is exhausted.

Figure 6.10 shows the performance and energy of the above adaptive sprint policy. 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.

6.5.3 A Gradual Sprint Pacing Policy

An alternative approach is to pace sprints based on instantaneous temperature. A sprint can be envisioned as the total computation performed as system temperature rises from an initial minimum (50°C) to a rated maximum (75°C). A temperature-based sprint pacing policy has the opportunity to

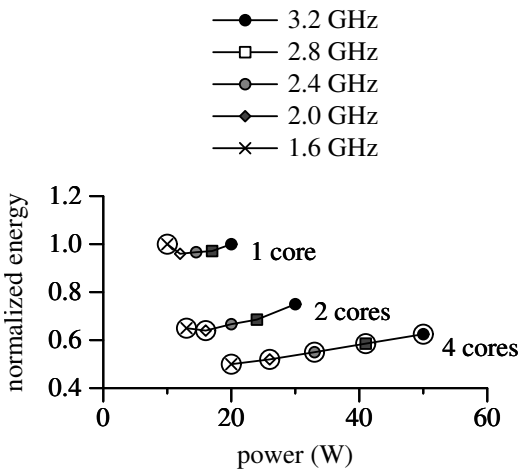


Figure 6.11: Circled power configurations are used for gradual sprint pacing.

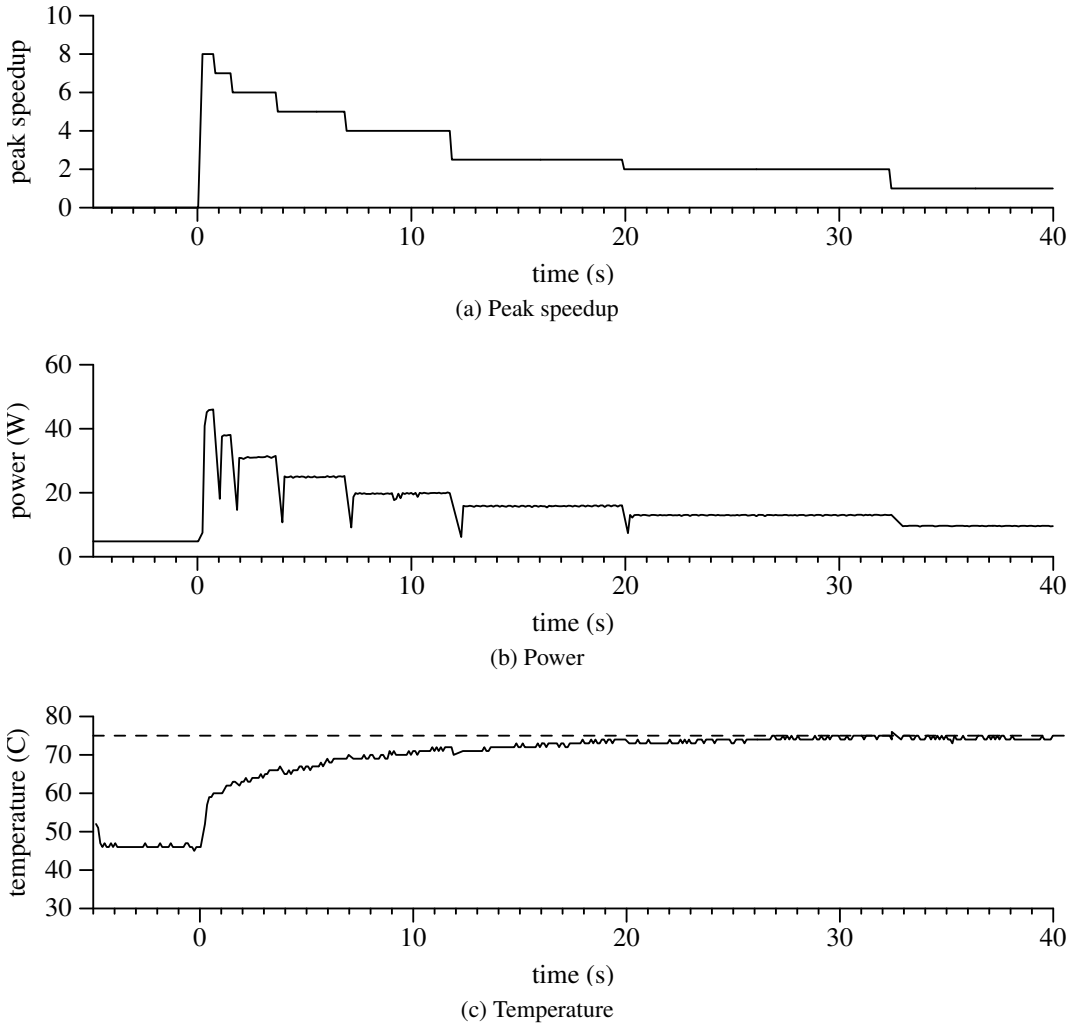
Temperature	**C**onfiguration	**P**ower
50-62	4 cores, 3.2 GHz	50.2 W
62-65	4 cores, 2.8 GHz	40.9 W
65-67	4 cores, 2.4 GHz	33.8 W
67-70	4 cores, 2.0 GHz	25.1 W
70-73	4 cores, 1.6 GHz	20.1 W
73-74	2 cores, 2.0 GHz	15.9 W
74-75	2 cores, 1.6 GHz	13.3 W
75-	1 core, 1.6 GHz	10 W

Table 6.3: Temperature-configuration settings for gradual sprint pacing.

influence the amount of computation performed by deciding the rate of computation at any instant based on the temperature and history. When viewed in this context, Figure 6.9 represents *static* policies which fix a constant sprint mode (and hence intensity) regardless of temperature.

An adaptive policy can vary the sprinting configuration based on temperature. Similar to the energy-based policy above, the goal of such a policy is to provide large responsiveness benefits across a wide length of computation. Hence the evaluated policy begins by computing with the maximum responsiveness (four cores, 3.2 GHz) configuration, and adaptively throttles execution through lower power configurations. To maximize performance, the policy traverses through lower-power configurations along the pareto-frontier marked by the circled points in Figure 6.11. Table 6.3 lists the temperature and sprinting configuration settings in the evaluated policy.

Figure 6.12a shows the instantaneous peak speedup as the system paces its sprint, throttling down from 8× when sprinting with all four cores at 3.2 GHz (50 W in Figure 6.12b) all the way to the 1× baseline over 30 s. Figure 6.12c shows the temperature of the system during the execution. The relatively smooth, monotonic rise in temperature highlights another advantage of sprint pacing; whereas abrupt sprint truncation results in an initial drop in temperature (recall from Figure 5.9b), gradually reducing power affords more time for heat to spread across the available thermal capac-



(a) Peak speedup

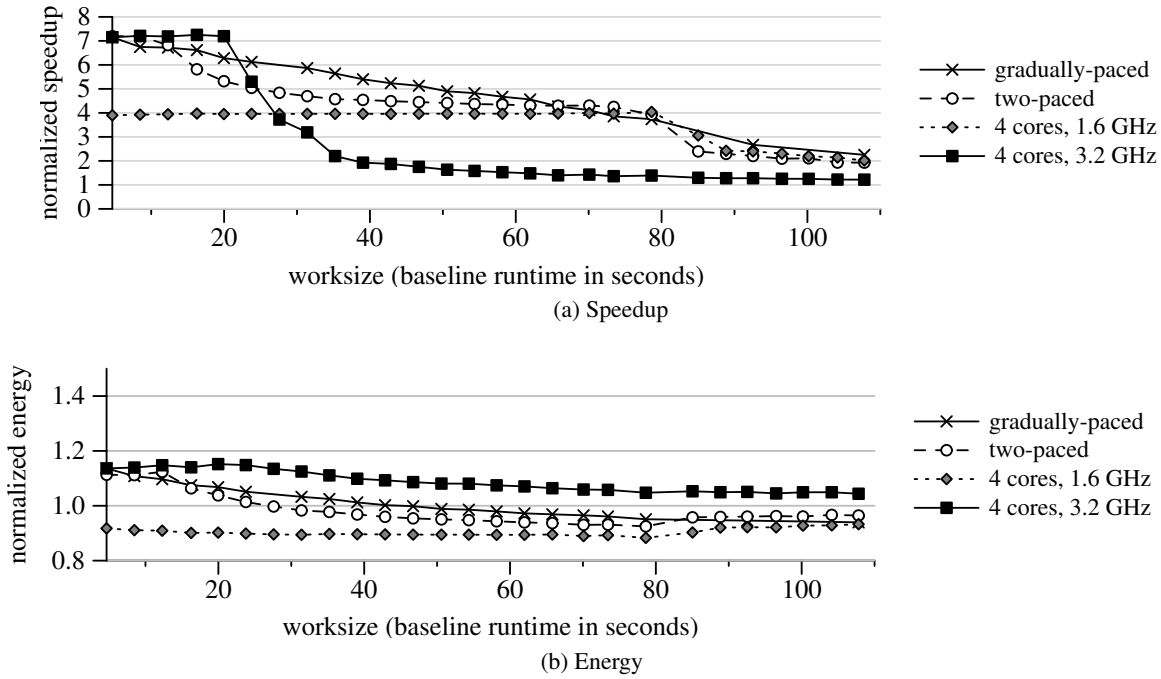
(b) Power

(c) Temperature

Figure 6.12: Power and temperature for temperature-based throttling.

itance and helps maintain the thermal equilibrium between the rate of heat generation and rate of heat dissipation.

Figure 6.13a and Figure 6.13b show the responsiveness and energy of the temperature-based pacing policy for the `sobel` workload. When the workload length is less than 3.8 s of baseline execution, all computation completes at maximum intensity (511 ms of sprinting). Because the most intense sprint can serve up to 23 s of baseline computation, the paced sprint sacrifices some performance (6× versus 7.3×) in favor of extending the responsiveness benefits to longer computations. As the computation length increases, the temperature-paced sprint outperforms static and



(a) Speedup

(b) Energy

Figure 6.13: Speedup and energy for temperature-based sprint pacing.

energy-paced sprinting, barring a short range (around 80 s of baseline computation) in which the four core, 1.6 GHz sprint is able to provide 4× speedup whereas the paced sprint transitions to the two core operating modes. For computations beyond this range, the temperature paced sprint is able to provide some speedup in excess of even the most energy efficient sprint (four core, 1.6 GHz) because more thermal capacitance is available when sprinting with lower intensity; this effect is visible as the excess energy used by the paced sprint in Figure 6.13b,

Overall, this experiment demonstrates that gradually throttling operating power extends the benefits of sprinting to a wider range of computation lengths than more abrupt forms of sprint truncation. The advantages of gradual sprinting are due to: (i) increased "effective" thermal capacitance due to more gradual heat spreading and (ii) adopting more energy-efficient sprinting configurations.

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 [100, 101, 127]. For example, a workload that has poor parallel scaling may benefit more

from higher frequency than additional cores. In the four-core testbed with workloads that scale well, such effects were found not to be significant, but they will likely become more critical in the future as the number of cores on a chip increases.

The above example implementations illustrate how a runtime system can gauge the thermal state of the system by monitoring temperature or energy, and choose a sprinting policy based on the current and previous thermal state. Prior work has explored similar techniques utilizing power to estimate temperature in the context of co-scheduling applications in heterogeneous CPU-GPU settings [129]. The Intel Linux Thermal Daemon (under development as of the time of this writing) is designed with a temperature throttling mechanism similar to the temperature-paced sprint above. To exploit headroom from thermal capacitance in future systems (*e.g.*, more aggressive TurboBoost), a sprint pacing policy could likely be coupled with such a service.

6.6 Sprint-and-Rest for Sustained Computations

The previous sections only analyzed sprint policy as a function of workload size while considering only a single sprint. However, the energy results from Section 6.2 and Section 6.3 suggest that sprinting can sometimes complete a given amount of computation using less energy than a thermally sustainable baseline mode even when considering a post-sprint idle period. This section hence investigates a repeated sprint-and-rest execution model as an alternative to the conventional steady rate of computation at TDP even for sustained workloads.

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.

6.6.1 Sprint-and-Rest Energy Analysis

Long-running computations are conventionally executed at a steady, sustainable operating mode that consumes less power than the rate at which the system can dissipate heat (allowing the chip to operate indefinitely). More directly, sprint-and-rest operation 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. If the thermal power limit of the system is $P_{sustainable}$ when operating with 1 core at f_{min}, then any increase of cores or frequency is therefore unsustainable and must only be engaged for a fraction of time $r_{sprint}(N,\ f)$ after which the system must idle:

$$\begin{aligned} P_{sprint\&rest}(N,\ f) &\leq P_{sustainable} \\ P_{total}(N,\ f) \cdot r_{sprint}(N,\ f) + P_{idle} \cdot (1 - r_{sprint}(N,\ f)) &\leq P_{sustainable} \\ \implies r_{sprint}(N,\ f) &\leq \frac{P_{sustainable} - P_{idle}}{P_{total}(N,\ f) - P_{idle}} \end{aligned} \tag{6.6}$$

As active computation only occurs in the sprint phase, the effective speedup of sprint-and-rest operation over steady baseline operation at $P_{sustainable}$ is:

$$\begin{aligned} S_{sprint\&rest}(N,\ f) &= S(N,\ f) \cdot r_{sprint}(N,\ f) \\ &\leq S(N,\ f) \cdot \frac{P_{sustainable} - P_{idle}}{P_{total}(N,\ f) - P_{idle}} \end{aligned} \tag{6.7}$$

The energy of executing in sprint-and-rest mode is:

$$E_{sprint\&rest}(N,\ f) = \frac{P_{sprint\&rest}(N,\ f)}{S_{sprint\&rest}(N,\ f)}$$

Therefore, when the sprint-rest power is exactly sustainable ($P_{sprint\&rest}(N,\ f) = P_{sustainable}$), the relative energy becomes:

$$Relative\ Energy = \frac{E_{sprint\&rest}(N,\ f)}{E_{compute}(1,\ f_{min})} = \frac{1}{S(N,\ f)} \cdot \frac{P_{total}(N,\ f) - P_{idle}}{P_{sustainable} - P_{idle}} \tag{6.8}$$

These inferences are next evaluated experimentally.

6.6.2 Evaluating Sprint-and-Rest

To contrast sprint-and-rest from conventional sustainable operation, consider (i) sustainable execution using a single-core at 1.6 GHz, (ii) sprint-and-rest operation with four cores at 1.6 GHz (energy

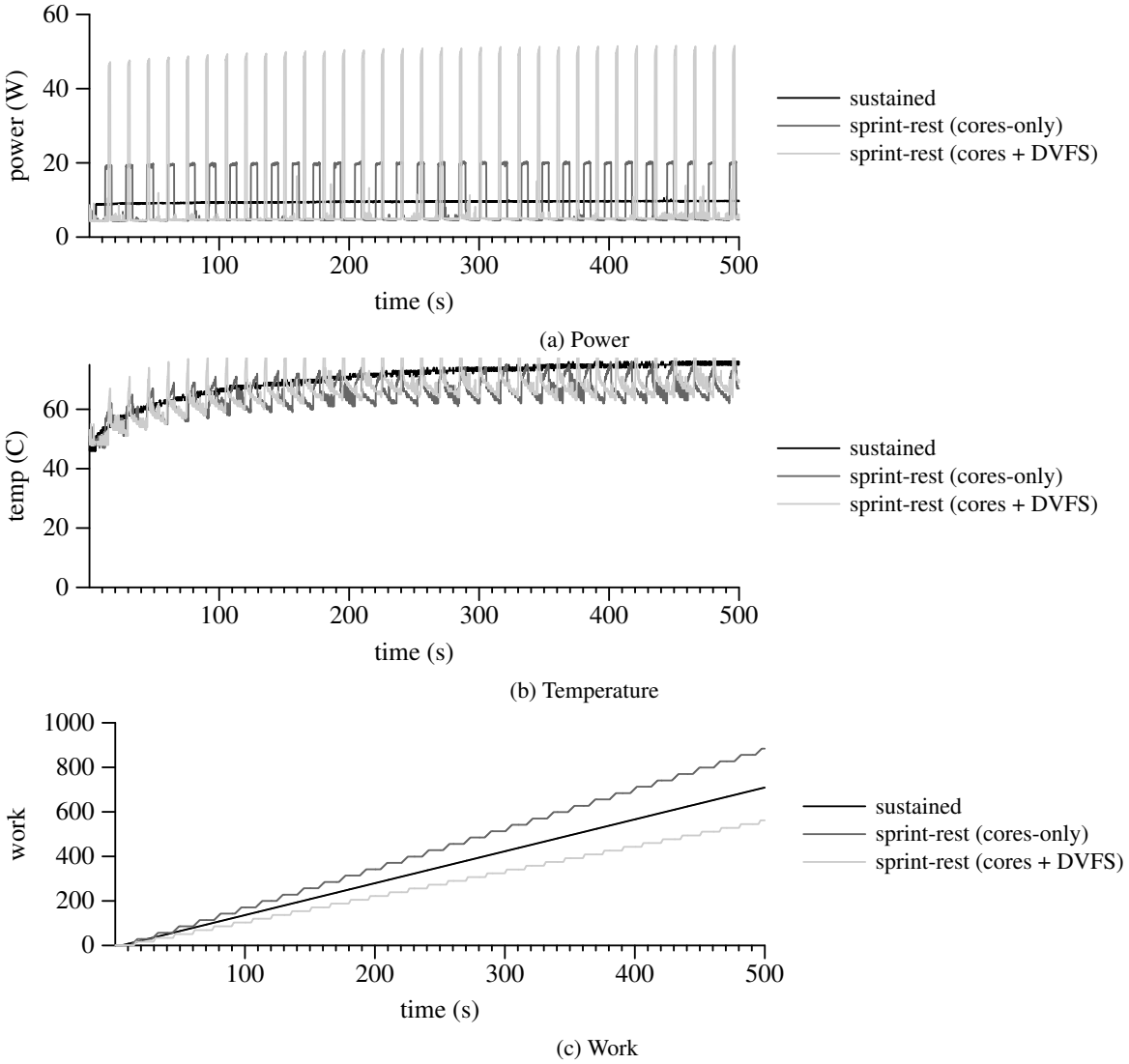


Figure 6.14: Comparison of power, temperarature, and cumulative work done with sprint-and-rest and sustained computation.

---

efficient configuration, referred to as `Parallel` sprinting), and (iii) sprint-and-rest operation with four cores at 3.2 GHz (called `Parallel+DVFS` sprinting).

To avoid overheating during an individual sprint, sprint duration for `Parallel` sprinting may not exceed 20 s. The evaluation selects a sprint duration of 5 s and a rest duration of 10.5 s ($r_{sprint}(N,\ f) = 1 : 3.1$). Similarly, for `Parallel+DVFS` sprinting at 50 W, the sprint duration is chosen as 1.5 s (less than the 3 s maximum sprint duration), and the corresponding rest

duration is 12.3 s ($r_{sprint}(N,\ f) = 1 : 9.1$). Substituting these values in Equation 6.8, `Parallel` sprinting is expected to be 23% more energy efficient, and `Parallel+DVFS` sprinting to be 22% less energy efficient compared to sustained execution at constant 10 W power.

Figure 6.14a and Figure 6.14b show the power traces, and corresponding thermal responses for the `sobel` workload executed on the testbed for over eight minutes with sustained and sprint-and-rest modes (for both `Parallel` and `Parallel+DVFS` sprinting) under the above duty cycles. The sawtooth temperature profiles as the system sprints and idles converge along 75°C, confirming that the average operating power is sustainable (10 W). Figure 6.14c compares the resulting cumulative work done when operating in these modes. The `Parallel+DVFS` sprint-and-rest mode under-performs sustained execution at the sustainable thermal limit by 21%. However, the `Parallel` mode of sprint-and-rest performs 20% more work over sustained operation on average.

The experiment confirms that sprint-and-rest provides a net efficiency gain when the instaneous energy-efficiency ratio of sprint versus 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. 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 (Section 4.4).

6.7 Chapter Summary

This chapter used the sprinting testbed to illustrate that sprinting can improve responsiveness and save energy by exposing higher-power configurations which are precluded by conventional TDP-constrained operation. The energy savings result from amortizing background power of uncore (*i.e.*, non-compute) components like the cache by (i) increasing compute resources (cores/frequency) to execute more efficiently (per-op energy), and (ii) finishing computation sooner and reducing the time for which these background components need to remain powered on. This concept of sprinting to save energy could well extend to future processors where the "dark-silicon" area devoted to the uncore is sized to support peak execution throughput (including sprinting with multiple otherwise dark cores). The energy benefits of sprinting further improve with reduction in idle energy. On the testbed system, even sprinting with frequency boosting is operationally more efficient than not

sprinting at all (23% on average); however, the idle energy is large enough to offset any such advantage and instead causes a net energy overhead (21%). In contrast, sprinting at minimum frequency with parallelism alone is more energy efficient (40% computational energy saving over baseline), resulting in net energy savings even accounting for idle power (average 6%).

This chapter also demonstrated a practical implementation of a software runtime to monitor and control sprinting. To ensure functional safety, the runtime truncates sprinting by throttling execution power based on thermal limits. Whereas naive sprint truncation by multiplexing active threads to a single core was shown to sometimes cause oversubscription penalties, a sprint-aware task-stealing framework is able to adaptively deactivate worker threads so that worst-case execution converges to sustainable baseline performance with only a single workload thread executing on the remaining core.

Having established a system where both frequency and core activity can be dynamically changed without loss in performance (compared to non-sprinting operation), further opportunity to improve sprinting performance lies in *pacing* sprints according to workload computation demand. The case for pacing sprints arises in the first place because the total computation that can be performed within the thermal headroom varies with sprint intensity. In the absence of *a priori* knowledge of the workload, a pacing policy can dynamically select sprint intensity based on the instantaneous thermal state of the system. Example implementations of both energy and temperature based policies showed that dynamically traversing from higher to lower sprint intensities along the pareto-optimal power-performance configurations helps capture the responsiveness benefits of sprinting over a wide range of computation lengths.

Finally, the energy benefits of sprinting can extend to long running workloads which far exceed the computation capacity of a single sprint. Alternating sprinting operation at an energy-efficient configuration (which exceeds TDP) with an idle period which is long enough to cool down the processor for a subsequent such sprint results in more computation performed at sustainable average power. Therefore, at least for some workloads, sprint-and-rest could potentially supplant even the baseline operating mode in systems where the thermally constrained operating configuration is not the most energy efficient.

Chapter 7

Conclusions

This chapter concludes this dissertation with a summary of the work presented in the preceding chapters (Section 7.1), directions for follow-up work (Section 7.2) and finally closes with my reflections on dark silicon, thermal design power and computational sprinting (Section 7.3).

7.1 Dissertation Summary

This dissertation proposed computational sprinting, an operating regime which exceeds sustainable power limits for intermittent bursts of intense computation by temporarily buffering heat using thermal capacitance. Computational sprinting was motivated by: (i) the widening gap (10×) in the estimated peak power-performance of future chips and the sustainable cooling limits of existing platforms, and (ii) the intermittent, rather than sustained computation demand of emerging interactive workloads.

Because conventional systems are not designed with unsustainable power supply, heat buffering or bursty performance in mind, this dissertation first investigated the challenges of sprinting under the severe thermal constraints projected in future systems finding that:

- Although the electrical supply demands of sprinting exceed the capability of existing mobile phone batteries, developments in battery and hybrid-ultracapacitor promise high-power, high-energy density supplies which can enable future sprint-capable systems.
- Three thermal parameters affect a system's ability to sprint: (i) rate of heat flow away from the die (thermal conductivity) determines maximum sprint intensity (power), (ii) thermal ca-

pacitance determines maximum duration for a sprint of a given intensity, (iii) thermal conductivity between the thermal capacitor and the surrounding ambient environment determines cool down time between sprints.

- Based on (i) and (ii) from above, this work proposed provisioning large sources of thermal capacitance close to the die to support high intensity sprints for up to one second. Experiments on a thermally constrained real system empirically validated this approach. The specific heat from the copper heat spreader inside a desktop chip allowed the system to exceed its sustainable power by 5× for 3 s.

- To further enhance heat buffering, this dissertation proposed exploiting the large latent heat of phase-change. Early experiments with a paraffin-infused heatsink prototype showed significant extensions for relative lower intensity sprints and pointed to the need for improved heat spreading to support maximum sprint intensity. Experiments also confirmed that adding thermal capacitance enhances sprint power, but also increases cooling latency (based on (iii) from above).

This dissertation then investigated the responsiveness and energy savings potential of sprinting, finding that:

- When the entire computation fits within the capacity of an unabridged sprint, the system exhibits the responsiveness of a thermally unconstrained system.

- Longer computations causing truncated sprints sometimes result in large penalties due to oversubscription. To ensure sprinting never results in worse performance than not sprinting, this dissertation proposed and evaluated a sprint-aware task-stealing framework.

- Despite higher total power, sprinting can save energy when compared to non-sprinting operation at sustainable power when: (i) overhead energy from powering background/uncore components are better amortized by higher compute power, resulting in energy savings per-operation during computation, and (ii) the energy saved during computation exceeds the idle energy of the system when it subsequently rests.

- The total computation possible under a single sprint was found to vary with sprint intensity. To avoid abrupt performance degradation, this dissertation proposed pacing sprints by dy-

namically stepping down operating power. Two example implementations showed that sprint pacing extended the responsiveness benefits of sprinting to a larger range of computation lengths and resulted in a more gradual degradation of performance than a static sprint policy.

- On thermally constrained systems where the sustainable baseline is not the most energy-efficient, repeated sprint-rest operation at unsustainable peak-power (but sustainable average power) outperformed the conventional model of steadily computing at the sustainable rate even for sustained workloads.

To conclude, this dissertation makes the case that signficant performance and energy benefits lie beyond the sustainable power margins of future mobile platforms. Engineering systems to temporarily break the TDP-bottleneck by sprinting at such unsustainable operating levels has the potential to enable entirely new interactive applications on devices several times more responsive than the state-of-the-art. This dissertation suggested a wholistic approach to engineering such a system, with an emphasis on the thermal and computing aspects. The next section outlines some directions for future work.

7.2 Directions for Future Work

This dissertation captures the initial findings of a larger collaborative project. Several practical questions remain to be answered towards realizing a reliable, cost-effective sprinting system. The group (comprising computer scientists, electrical engineers, and mechanical engineers at the University of Pennsylvania and the University of Michigan) continues to investigate some of the following questions.

7.2.1 How Does Sprinting Impact Reliability?

Section 4.7 pointed to the reliability concerns raised by the cyclical nature of sprinting operation such as fatigue and wear from thermal stresses on the processor and solder bumps, electromigration, and integrity of phase-change materials. Although sprinting never exceeds safe temperature limits, the choice of phase-change temperature can impact reliability by regulating the average temperature of the processor (for example, a lower melting point would keep the processor cooler at the cost of increased thermal cycling of the PCM itself). Such operation suggests that

the failure modes facilitated by sprinting (if any) would more likely arise from gradual aging (and degradation of performance), rather than abrupt failure. This dissertation makes the observation that desktop-class chips commonly encounter the estimated sprinting conditions; however, future technology nodes have raised variation and reliability concerns even for desktop and server class chips [20, 84, 158]. Understanding whether and how sprinting impacts reliability likely requires deep study.

7.2.2 What is the Metric of Sprinting Performance?

This dissertation largely focused on evaluating *single* sprints, in which traditional measures such as speedup or runtime capture relative performance. However, system performance across repeated sprints would vary depending on usage patterns (such as computation task lengths and distribution of active-vs-idle periods). Benchmarking and measurement metrics are hence required to compare the relative merits of sprinting systems. The architecture community has similarly revisited performance metrics when faced with new execution paradigms in the past; for example, the rising popularity in multiprocessor architecture led to a re-evaulation of performance metrics to accommodate parallel computation [10, 99].

7.2.3 How Does Sprinting Generalize Beyond Parallelism and Frequency Boosting?

Mobile chips today already incorporate multiple cores, GPUs, and a dozen or so specific function units, not all of which can be sustained simultaneously. The limits of passive convection extend to all such components on chip, and not just the cores. Although this dissertation focused on parallelism and voltage-frequency boosting using the cores, the headroom from thermal capacitance is agnostic to *how* it is used. Thus, there may be a case for sprinting on GPUs or accelerators in addition to core and frequency-based sprinting. For example, recent work explored trading-off active power between the on-die CPU and GPU by exploiting the lay-out dependent thermal coupling between them [129].

7.2.4 When is Sprinting Useful?

As seen in Section 4.3, sprinting performance is constrained by utilization—a system which sprints by a factor of S over TDP can only be utilized $\frac{1}{S}$ of the total time on average (assuming energy-

proportional performance). This dissertation largely focused on order-of-magnitude sprints—10× in excess of TDP for a few seconds—aiming at interactive applications, and considered mobile device constraints. However, one can also consider the impact of sprinting under entirely different scenarios, some of which are discussed below:

Less intense sprinting for server applications. While making their case for energy-proportional computing in the server space, Barroso and Hölzle note [15]:

"Mobile devices require high performance for short periods while the user awaits a response, followed by relatively long idle intervals of seconds or minutes...servers are rarely completely idle and seldom operate near their maximum utilization. Instead, servers operate most of the time at between 10 and 50 percent of their maximum utilization levels."

The same article goes on to note that servers are routinely over-provisioned to cater to occasions of peak utilization. Based on these observations, it might be attractive to reduce over-provisioning costs by sprinting in response to such periods of heavy utilization. The feasibility studies in this dissertation—which consider power in the range of tens of watts—would need to be reproduced in the context of servers which are order-of-magnitude more power hungry. However, mechanisms like TurboBoost 2.0 suggest that it may be more readily feasible to engineer for modest sprints to handle bursty loads—for example exceeding TDP by a factor of two for minutes.

Scenarios where sprinting may not be useful. Applications such as video games, streaming media and batch-mode scientific computing can require sustained, high-performance computation for long durations (lasting hours and even days). Such applications are more likely to benefit from energy-efficient, dedicated architectures (like accelerators, special function units) than relying on sprint-and-rest to harvest performance out of less energy-efficient architectures.

7.2.5 How should Applications Utilize Thermal Headroom?

This dissertation made the case for pacing even a single sprint to adapt to the computation task. In a potential future scenario where multiple applications contend for the thermal headroom, there may be a case for extending conventional resource-allocation and management services, for example in the operating system or hypervisor. User studies could point to patterns of idleness and computation conducive to sprinting. Previous user studies have demonstrated user discontent with degrading performance in interactive scenarios [79, 152]; an extension of such a study gathering how users

react to the performance of a "tiring" phone can further influence sprint pacing. "Computational re-sprinting" [164], analyzes repeated sprinting under quality-of-service constraints.

7.3 Reflections on Power and Thermal Constraints

This section reflects some of my opinions on process scaling trends and their implications on client-side computation.[1] These opinions formed over the course of my academic research, fostered by several conversations with colleagues, a summer internship, and prior employment in the computer industry. I advocate for task-based parallelism as a "performance-robust" execution model in systems with variable performance (Section 7.3.1), expect energy efficiency to increasingly favor sprint-to-idle (Section 7.3.2), and call for a redefinition of TDP (Section 7.3.3).

7.3.1 Task-parallelism is a Unifying, Performance-Robust Framework.

Task-based frameworks [3, 24] combine several desirable properties of a flexible, performance-robust operating paradigm. In this dissertation, the thread-oblivious nature allowed dynamic core activation/deactivation without affecting performance adversely. Task-parallel frameworks have fairly low runtime overheads and are amenable to load-balancing by task-stealing [21, 39]. Previous work has explored hardware support for reducing task runtime overheads [95], marshaling data when tasks migrate cores (including heterogenous contexts) [160], and task scheduling decisions [171]. Task-based frameworks provide a generic, shared memory programming interface to adapt to variable performance "sprint-capable" platforms in future.

7.3.2 Design for Sprint-to-Idle

Although we initially envisioned sprinting as a performance/responsiveness mechanism, the energy implications may yet make a stronger case for sprinting.

- **Mobile devices already design for low power idle.** Most mobile chips today optimize for the extremes of low-power idle and high-performance active operation [15] modes, such as NVIDIA's variable-SMP and ARM's big.LITTLE architectures [61, 124]. As seen in Section 6.3, sprinting benefits from lower idle power.

[1] I use the first person singular "I" for this section alone because these thoughts may not necessarily be shared by my co-authors of the published papers.

- **Background power will grow with many-cores.** The above multi-core chips have so far sized the shared L2 cache at 256-512KB per-core (*i.e.*, total cache sizes of 512KB-2MB for dual and quad-core chips). If the trend of increasing core counts continues, it is conceivable for future chips to contain caches comparable to today's desktop chips—a cause for increased background energy. (For example, the 8MB shared L3 contributed to the background power of the testbed processor used in this dissertation). This background energy is a "necessary evil" to extract performance from the cores—Borkar suggests that processors were cache-starved when transistors were predominantly allocated to core performance [27]; Hill and Wood's "costup" argument of investing in "non-compute" resources follows similar reasoning [177].

Both the above factors—idle power and background power—look to be trending towards race-to-idle as the more energy efficient operating point (as per the analysis in Section 6.3). If the operating power of the performance-optimized cores grows to be unsustainable following along the estimated trends, then using these cores in sprint mode could in fact be the most energy-efficient way to utilize the system. In the words of Greg Wright, an engineer from Qualcomm:

We're taught from being kids that slow and steady ought to win the race; the tortoise deserves to beat the hare...The tortoise is out there with the sun beating down on his shell, while the hare is cooling off under a tree. As static power becomes increasingly significant, the optimal design point shifts towards sprint and rest.

7.3.3 Revisit TDP as the Scope of Thermal Constraints

Previously published sources [47, 169], including the one used in Chapter 2 estimate utilization as the shrinking complement of growing power density. Hence, the utilization wall and dark silicon trends have been estimated based on sustainable cooling constraints alone. The rationale for limiting power to reduce battery load and hence extend battery life [111] is less compelling in light of recent high-discharge supplies—although lower *average power* is still desirable to extend battery life, the high-C discharge rate resulting from intermittent *peak-power* operation is unlikely to reduce the energy capacity of future power supplies.

Even in the absence of worsening CMOS power, the poor heat transfer from natural convection (Section 4.1) can restrict sustainable power in the range of 1-2 W (and even lower if the ambient

environment is warm). Our experiments with recent phones showed that once the case temperature neared tolerable margins, the phone started to throttle even under today's continuous loads (although it took over a half hour for the case to heat up at the low wattage of today's phones). Our conversations with researchers and engineers from leading semiconductor and mobile vendors suggested a growing realization of the need for an alternative focus to sustainable operation.

Thermal capacitance essentially breaks a single bottleneck heat channel (thermal resistance) into segments connecting thermal capacitors. The capacitors closest to the die are expectedly the smallest, yet quickest to absorb heat (low latency of heat transfer, low thermal capacitance), with successively farther components increasing in thermal resistance and capacitance. This suggests building a thermal hierarchy much like a memory hierarchy, in which the smallest fastest registers and caches are closest to the computation units and larger caches are slower to access).

Computer architecture and microarchitecture have maintained a singular focus on lower-power designs for the mobile domain. Relaxing TDP constraints can enable hardware mechanisms which may otherwise be summarily rejected due to unsustainable power (for example, deep speculation). In a regime of sprint-and-rest where higher performance translates to increased energy-efficiency, the energy costs of added performance could well pay for themselves. My concluding thoughts from this work are that *when TDP does not constrain performance, dark silicon is not a concern.*

Bibliography

[1] ACPI Specification v0.91. URL http://www.intel.com/content/www/us/en/standards/acpi-specification-v091.html.

[2] Linux Thermal Daemon. URL https://01.org/linux-thermal-daemon/documentation/introduction-thermal-daemon.

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

[4] Failure Mechanisms and Models for Semiconductor Devices. *JEDEC Publication*, (122C), 2006.

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

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

[7] International Technology Roadmap for Semiconductors, 2010 update, 2010. URL http://www.itrs.net.

[8] 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.

[9] Intel®64 and IA-32 Software Developer's Manual, Volume 3B: System Programming Guide, Part 2, Aug. 2012.

[10] A. Alameldeen and D. A. Wood. IPC Considered Harmful for Multiprocessor Workloads. *IEEE MICRO*, 26(4):8–17, 2006.

[11] E. Alawadhi and C. Amon. Thermal Analyses of a PCM Thermal Control Unit for Portable Electronic Devices: Experimental and Numerical Studies. *IEEE Trans. on Components and Packaging Technology*, 26:116–125, 2003.

[12] 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*, pages 1266–1285, 2012.

[13] 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.

[14] 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.

[15] L. Barroso and U. Holzle. The Case for Energy-Proportional Computing. *Computer*, 40, 2007.

[16] F. Baskett, K. M. Chandy, R. R. Richar R. Muntz, and F. G. Palacios. Open, Closed, and Mixed Networks of Queues with Different Classes of Customers. *Journal of the ACM*, 22 (2):248–260, Apr 1975.

[17] J. V. Bellemare. Thermally Reflective Encapsulated Phase Change Pigment.

[18] M. Berktold and M. Tian. CPU Monitoring with DTS/PECI. *Intel White Paper*, Sep 2009.

[19] T. E. Bernard and M. F. Foley. Upper Acceptable Surface Temperature for Prolonged Hand Contact. *International Journal of Industrial Ergonomics*, 11(1):29–36, 1993.

[20] K. Bernstein, D. J. Frank, A. E. Gattiker, W. Haensch, B. L. Ji, S. R. Nassif, E. J. Nowak, D. J. Pearson, and N. J. Rohrer. High-Performance CMOS Variability in the 65-nm Regime and Beyond. *IBM Journal of Research and Development*, 50(4/5):433–449, Jul 2006.

[21] A. Bhattacharjee and M. Martonosi. Thread Criticality Predictors for Dynamic Performance, Power, and Resource Management in Chip Multiprocessors. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, June 2009.

[22] 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.

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

[24] 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.

[25] M. Bohr. A 30 Year Retrospective on Dennard's MOSFET Scaling Paper. *IEEE Solid-State Circuits Society Newsletter*, 12(1):11–13, 2007.

[26] S. Borkar. Major Challenges to Achieve Exascale Performance. *Salishan Conf. on High-Speed Computing*, 2009.

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

[28] R. J. Brodd, K. R. Bullock, R. A. Leising, R. L. Middaugh, J. R. Miller, and E. Takeuchi. Batteries, 1977 to 2002. *Journal of The Electrochemical Society*, 151(3):K1–K11, 2004.

[29] D. Brooks and M. Martonosi. Dynamic Thermal Management for High-Performance Microprocessors. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, July 2001.

[30] L. Cao, J. P. Krusius, M. A. Korhonen, and T. S. Fisher. Transient Thermal Management of Portable Electronics Using Heat Storage and Dynamic Power Dissipation Control. *IEEE Trans. on Components, Packaging, and Manufacturing Technology*, 21(1), Mar. 1998.

[31] K. Chakraborty, P. M. Wells, and G. S. Sohi. Computational Spreading: Employing Hardware Migration to Specialze CMP Cores On-the-fly. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 2006.

[32] A. P. Chandrakasan, W. J. Bowhill, and F. Fox. *Design of High-Performance Microprocessor Circuits*. Wiley-IEEE Press, 1st edition, 2000.

[33] T. Chantem, X. S. Hu, and R. P. Dick. Online Work Maximization Under a Peak Temperature Constraint. In *Proceedings of the 2009 International Symposium on Low Power Electronics and Design*, 2009.

[34] 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/`.

[35] C.-P. Chiu, G. L. Solbrekken, V. LeBonheur, and Y. E. Xu. Application of Phase-Change Materials in Pentium III and Pentium III Xeon TM Processor Cartridges. *International Symposium on Advanced Packaging Materials*, 2000.

[36] B. Choi, L. Porter, and D. Tullsen. Accurate Branch Prediction for Short Threads. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2008.

[37] J. Clemons, A. J. R. Perricone, and a. T. A. Silvio Savarese. EFFEX: An Embedded Processor for Computer Vision Based Feature Extraction. In *Proceedings of the 48th Design Automation Conference*, June 2011.

[38] 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.

[39] 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.

[40] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. MAUI: making smartphones last longer with code offload. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, 2010.

[41] R. H. Denard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. Leblanc. Design of Ion-implanted MOSFETs with Very Small Physical Dimensions. *IEEE Journal of Solid-state Circuits*, 98, 1974.

[42] W. Doherty and A. Thadhani. The economic value of rapid response time. Technical report, IBM, 1982.

[43] J. Donald and M. Martonosi. Techniques for Multicore Thermal Management: Classification and New Exploration. June 2006.

[44] R. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge. Near-Threshold Computing: Reclaiming Moore's Law Through Energy Efficient Integrated Circuits. *Proceedings of the IEEE*, 98(2):253–266, 2010.

[45] B. Erol, E. Antunez, and J. J. Hull. HOTPAPER: Multimedia Interaction Interaction with Paper using Mobile Phones. In *Proceedings of the International Symposium on Multimedia*, 2008.

[46] 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.

[47] 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.

[48] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. Neural Acceleration for General-Purpose Approximate Programs. In *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture*, Nov. 2012.

[49] H. M. Ettouney, T. A. I, and S. A. A.-A. S. M. Al-Sahali. Heat Transfer Enhancement by Metal Screens and Metal Spheres in Phase Change Energy Storage Systems. *Renewable Energy*, 29, 2004.

[50] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin. Diversity in smartphone usage. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, 2010.

[51] D. Fick, R. Dreslinski, B. Giridhar, G. Kim, S. Seo, M. Fojtik, S. Satpath, Y. Lee, D. Kim, N. Liu, M. Wieckowski, G. Chen, T. Mudge, D. Sylvester, and D. Blaauw. Centip3De: A 3930DMIPS/W configurable Near-Threshold 3D Stacked System with 64 ARM Cortex-M3 cores. In *2012 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, 2012.

[52] K. Flautner, S. K. Reinhardt, and T. N. Mudge. Automatic Performance-Setting for Dynamic Voltage Scaling. In *Proceedings of the 7th Conference on Mobile Computing and Networking MOBICOM*, pages 260–271, 2001.

[53] 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.

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

[55] 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*, pages 157–168, June 2009.

[56] L. Gao, R. Dougal, and S. Liu. Power Enhancement of an Actively Controlled Battery/Ultracapacitor Hybrid. *IEEE Trans. on Power Electronics*, 20(1):236 – 243, Jan. 2005.

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

[58] 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.

[59] M. Glavin and W. Hurley. Ultracapacitor/Battery Hybrid for Solar Energy Storage. In *42nd International Universities Power Engineering Conference*, pages 791–795, Sept. 2007.

[60] N. Goulding-Hotta, J. Sampson, G. Venkatesh, S. Garcia, J. Auricchio, P.-C. Huang, M. Arora, S. Nath, V. Bhatt, J. Babb, S. Swanson, and M. B. Taylor. The GreenDroid Mobile Application Processor: An Architecture for Silicon's Dark Future. *IEEEMicro*, 21(2), 2011.

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

[62] S. H. Gunther, F. Binns, D. M. Carmean, and J. C. Hall. Managing the Impact of Increasing Microprocessor Power Consumption. *Intel Technology Journal*, Q1 2001.

[63] 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.

[64] S. P. Gurrum, Y. K. Joshi, and J. Kim. Thermal Management of High Temperature Pulsed Electronics Using Metallic Phase Change Materials. *Numerical Heat Transfer, Part A: Applications: An International Journal of Computation and Methodology Issue 8*, 42:777–790, 2002.

[65] A. Gutierrez, R. Dreslinski, T. F. Wenisch, T. Mudge, A. Saidi, C. Emmons, and N. Paver. Full-System Analysis and Characterization of Interactive Smartphone Applications. In *Proceedings of the IEEE International Symposium on Workload Characterization*, Sept. 2011.

[66] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz. Understanding Sources of Inefficiency in General-purpose Chips. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, June 2010.

[67] A. Hartl, L. Gruber, C. Arth, and D. Stefan Hauswiesner. Rapid Reconstruction of Small Objects on Mobile Phones. In *Proceedings of Seventh IEEE Workshop on Embedded Computer Vision*, 2011.

[68] M. Hodes, R. D. Weinstein, S. J. Pence, J. M. Piccini, L. Manzione, and C. Chen. Transient Thermal Management of a Handset Using Phase Change Material (PCM). *Journal of Electronic Packaging*, 124(4):419–426, 2002.

[69] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. Stan. HotSpot: A Compact Thermal Modeling Methodology for Early-stage VLSI Design. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 14(5):501–513, 2006.

[70] W. Huang, K. Rajamani, M. R. Stan, and K. Skadron. Scaling with Design Constraints: Predicting the Future of Big Chips. *IEEE Micro, IEEE*, 31(4):16–29, 2011.

[71] W. Huang, K. Skadron, S. Gurumurthi, R. Ribando, and M. Stan. Exploring the Thermal Impact on Manycore Processor Performance. In *Proceedings of the Twenty Sixth Annual IEEE Semiconductor Thermal Measurement and Management Symposium*, 2010.

[72] W. Huang, M. R. Stan, K. Sankaranarayanan, R. J. Ribando, and K. Skadron. Many-Core Design from a Thermal Perspective. In *Proceedings of the 45th Design Automation Conference*, June 2008.

[73] 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*, pages 254–265, July 2001.

[74] C. Isci, G. Contreras, and M. Martonosi. Live, Runtime Phase Monitoring and Prediction on Real Systems with Application to Dynamic Power Management. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2006.

[75] C. Isci and M. Martonosi. Runtime Power Monitoring in High-End Processors: Methodology and Data. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2003.

[76] S. Jain, S. Khare, S. Yada, V. Ambili, P. Salihundam, S. Ramani, S. Muthukumar, M. Srinivasan, and A. Kumar. A 280mV-to-1.2 V Wide-Operating-Range IA-32 Processor in 32nm CMOS. In *2012 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, 2012.

[77] M. Jaworski. Thermal Performance of Heat Spreader for Electronics Cooling with Incorporated Phase Change Material. *Applied Thermal Engineering*, 35, 2012.

[78] 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.

[79] S. Jorg, A. Normoyle, and A. Safonova. How Responsiveness Affects Players Perception in Digital Games. In *Proceedings of the ACM Symposium on Applied Perception*, 2012.

[80] D.-C. Juan, Y.-T. Chen, M.-C. Lee, and S.-C. Chang. An Efficient Wake-Up Strategy Considering Spurious Glitches Phenomenon for Power Gating Designs. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 18(2):246 –255, Feb. 2010.

[81] H. Jung and M. Pedram. Optimizing the Power Delivery Network in Dynamically Voltage Scaled Systems with Uncertain Power Mode Transition Times. In *Proceedings of the Conference on Design, Automation and Test in Europe*, Mar. 2010.

[82] R. Kandasamy, X.-Q. Wang, and A. S. Mujumdar. Application of Phase Change Materials in Thermal Management of Electronics. *Applied Thermal Engineering*, 27(17-18):2822 – 2832, 2007.

[83] B. Kang and G. Ceder. Battery Materials for Ultrafast Charging and Discharging. *Nature*, 458(7235):190–193, Mar. 2009.

[84] U. R. Karpuzcu, B. Greskamp, and J. Torrellas. The BubbleWrap Many-Core: Popping Cores for Sequential Acceleration. In *Proceedings of the 42nd International Symposium on Microarchitecture*, Nov. 2009.

[85] S. Kaxiras and M. Martonosi. Computer Architecture Techniques for Power-Efficiency. *Synthesis Lectures on Computer Architecture*, 3(1):1–207, 2008.

[86] J. H. Kelm, D. R. Johnson, M. R. Johnson, N. C. Crago, W. Tuohy, A. Mahesri, S. S. Lumetta, M. I. Frank, and S. J.Patel. Rigel: an Architecture and Scalable Programming Interface for a 1000-core Accelerator. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, June 2009.

[87] N. S. Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan. Leakage Current: Moore's Law Meets Static Power. *Computer*, 36(12): 68–75, Dec 2003.

[88] S. Kim, S. V. Kosonocky, and D. R. Knebel. Understanding and Minimizing Ground Bounce during Mode Transition of Power Gating Structures. In *Proceedings of the 2003 International Symposium on Low Power Electronics and Design*, 2003.

[89] W. Kim, M. S. Gupta, G.-Y. Wei, and D. Brooks. Enabling On-Chip Switching Regulators for Multi-Core Processors using Current Staggering. In *Proceedings of the Workshop on Architectural Support for Gigascale Integration*, 2007.

[90] W. Kim, M. S. Gupta, G.-Y. Wei, and D. Brooks. System Level Analysis of Fast, Per-Core DVFS Using On-Chip Switching Regulators. In *Proceedings of the 14th Symposium on High-Performance Computer Architecture*, Feb. 2008.

[91] J. Kong, S. W. Chung, and K. Skadron. Recent Thermal Management Techniques for Microprocessors. *ACM Computing Surveys (CSUR)*, 44(3):13, 2012.

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

[93] E. Krimer, R. Pawlowski, M. Erez, and P. Chiang. Synctium: A Near-threshold Stream Processor for Energy-constrained Parallel Applications. *Computer Architecture Letters*, 9 (1):21–24, 2010.

[94] A. Kumar, L. Shang, L.-S. Peh, and N. K. Jha. System-Level Dynamic Thermal Management for High-Performance Microprocessors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(1):96–108, 2008.

[95] S. Kumar, C. J. Hughes, and A. Nguyen. Carbon: Architectural Support for Fine-Grained Parallelism on Chip Multiprocessors. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, June 2007.

[96] A. Kuperman and I. Aharon. Battery-ultracapacitor Hybrids for Pulsed Current Loads: A Review. *Renewable and Sustainable Energy Reviews*, 15(2):981 – 992, 2011.

[97] E. Kursun and C. Y. Cher. Variation-aware Thermal Characterization and Management of Multi-core Architectures. In *Proceedings of the International Conference on Computer Design*, Oct. 2008.

[98] D. Lea. A Java Fork/Join Framework. In *Proceedings of the ACM Java Grande 2000 Conference*, pages 36–43, 2000.

[99] K. Lepak, H. Cain, and M. Lipasti. Redeeming IPC as a Performance Metric for Multithreaded Programs. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, 2003.

[100] 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.

[101] 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.

[102] J. Li, J. F. Martinez, and M. C. Huang. The Thrifty Barrier: Energy-Aware Synchronization in Shared-Memory Multiprocessors. In *Proceedings of the Tenth Symposium on High-Performance Computer Architecture*, Feb. 2004.

[103] M.-L. Li, R. Sasanka, S. Adve, Y.-K. Chen, and E. Debes. The ALPBench Benchmark Suite for Complex Multimedia Applications. Sept. 2005.

[104] P.-C. Li and T. K. Young. Electromigration: The Time Bomb in Deep-Submicron ICs. *IEEE Spectrum*, 33(9):75–78, 1996.

[105] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. In *Proceedings of the 42nd International Symposium on Microarchitecture*, Nov. 2009.

[106] 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, pages 271–283, Oct. 2004.

[107] Y. Li, B. Lee, D. Brooks, Z. Hu, and K. Skadron. Impact of Thermal Constraints on Multi-core Architectures. In *Proceedings of the Tenth Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronics Systems*, 2010.

[108] 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.

[109] 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.

[110] Z. Luo, H. Cho, X. Luo, and K. il Cho. System Thermal Analysis for Mobile Phone. *Applied Thermal Engineering*, 28(14-15):1889 – 1895, 2008.

[111] T. L. Martin. *Balancing Batteries, Power, and Performance: System Issues in CPU Speed-Setting for Mobile Computing*. PhD thesis, CARNEGIE MELLON UNIVERSITY, 1999.

[112] 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.

[113] 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-`.

[114] 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.

[115] F. J. Mesa-Martinez, J. Nayfach-Battilana, and J. Renau. Power Model Validation Through Thermal Measurements. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, June 2007.

[116] C. C. Minh, J. Chung, C. Kozyrakis, and K. Olukotun. STAMP: Stanford Transactional Applications for Multi-Processing. In *Proceedings of the IEEE International Symposium on Workload Characterization*, Sept. 2008.

[117] 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.

[118] 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*, pages 35–44, June 2002.

[119] M. E. Mngomezulu, A. S. Luyt, and I. Krupa. Structure and Properties of Phase Change materials Based on HDPE, Soft Fischer-Tropsch Paraffin Wax, and Wood Flour. *Journal of Applied Polymer Science*, 118:1541–1551, 2010.

[120] M. Monchiero. Design Space Exploration for Multicore Architectures: A Power/Performance/Thermal View. In *Proceedings of the 20th International Conference on Supercomputing*, June 2006.

[121] G. E. Moore. Cramming More Components Onto Integrated Circuits. *Electronics Magazine*, 38(8), 1965.

[122] T. G. Mudge. Power: A First-Class Architectural Design Constraint. *Computer*, 34(4):52–58, Apr 2001.

[123] R. Mukherjee and S. O. Memik. Physical Aware Frequency Selection for Dynamic Thermal Management in Multi-Core Systems. In *Proceedings of the International Conference on Computer Aided Design*, Nov. 2006.

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

[125] U. S. D. of Energy. Office of basic Energy Sciences. *Basic Research Needs for Electrical Energy Storage: Report of the Basic Energy Sciences Workshop on Electrical Energy Storage, April 2-4, 2007*. Office of Basic Energy Sciences, Department of Energy, 2007.

[126] 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, pages 918 – 923, June 2003.

[127] 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*, pages 169–180, June 2007.

[128] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang. Enabling Automatic Offloading of Resource-Intensive Smartphone Applications. *Purdue ECE Technical Reports. Paper 419*, 2011. URL `http://docs.lib.purdue.edu/ecetr/419`.

[129] I. Paul, S. Manne, M. Arora, W. L. Bircher, and S. Yalamanchili. Cooperative Boosting: Needy Versus Greedy Power Management. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, June 2013.

[130] 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.

[131] J. H. Pikul, H. G. Zhang, J. Cho, P. V. Braun, and W. P. King. High-power Lithium Ion Microbatteries from Interdigitated Three-Dimensional Bicontinuous Nanoporous Electrodes. *Nature Communications*, 4, 2013.

[132] M. D. Powell, M. Gomaa, and T. N. Vijaykumar. Heat-and-Run: Leveraging SMT and CMP to Manage Power Density Through the Operating System. In *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 2004.

[133] M. D. Powell and T. N. Vijaykumar. Resource Area Dilation to Reduce Power Density in Throughput Servers. In *Proceedings of the 2007 International Symposium on Low Power Electronics and Design*, 2007.

[134] K. Puttaswamy and G. H. Loh. Thermal Herding: Microarchitecture Techniques for Controlling Hotspots in High-Performance 3D-Integrated Processors. In *Proceedings of the 13th Symposium on High-Performance Computer Architecture*, Feb. 2007.

[135] A. Raghavan, L. Emurian, L. Shao, M. Papaefthymiou, K. P. Pipe, T. F. Wenisch, and M. M. K. Martin. Computational Sprinting on a Hardware/Software Testbed. In *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2013.

[136] A. Raghavan, L. Emurian, L. Shao, M. Papaefthymiou, K. P. Pipe, T. F. Wenisch, and M. M. K. Martin. Utilzing Dark Silicon to Save Energy with Computational Sprinting. *IEEE MICRO Special Issue on Dark Silicon*, 2013.

[137] 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.

[138] A. Raghavan, Y. Luo, A. Chandawalla, M. C. Papaefthymiou, K. P. Pipe, T. F. Wenisch, and M. M. K. Martin. Designing for Responsiveness with Computational Sprinting. *IEEE MICRO Top Picks in Computer Architecture of 2012*, 33(3), May-Jun 2013.

[139] R. Reddy. CMU Sphinx. URL `http://www.speech.cs.cmu.edu/sphinx`.

[140] A. Rogers, D. Kaplan, E. Quinnell, and B. Kwan. The Core-C6 (CC6) Sleep State of the AMD Bobcat x86 Microprocessor. In *Proceedings of the 2012 International Symposium on Low Power Electronics and Design*, 2012.

[141] 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.

[142] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand. Leakage Current Mechanisms and Leakage Reduction Techniques in Deep-Submicrometer CMOS Circuits. *Proceedings of the IEEE*, 91(2):305–327, 2003.

[143] P. B. Salunkhe and P. S. Shembekar. A Review on Effect of Phase Change Material Encapsulation on the Thermal Performance of a System. *Renewable and Sustainable Energy Reviews*, 16(8):5603–5616, 2012.

[144] A. A. Sani, W. Richter, X. Bao, T. Narayan, M. Satyanarayanan, L. Zhong, and R. R. Choudhury. Opportunistic Content Search of Smartphone Photos. *Technical Report TR0627-2011, Rice University*, 2011.

[145] A. Sari. Form-stable Paraffin/High Density Polyethylene Composites as SolidLiquid Phase Change Material for Thermal Energy Storage: Preparation and Thermal Properties. *Energy Conversion and Management*, 45, 2004.

[146] J. Schindall. The Charge of the Ultracapacitors. *IEEE Spectrum*, 44(11):42–46, Nov. 2007.

[147] E. Schurman and J. Brutlag. The User and Business Impact of Server Delays, Additional Bytes, and HTTP Chunking in Web Search. *Velocity*, 2009.

[148] G. Setoh, F. Tan, and S. Fok. Experimental Studies on the use of Phase Change Material for Cooling Mobile Phones. *International Communications in Heat and Mass Transfer*, 37(9): 1403 – 1410, 2010.

[149] L. Shao, A. Raghavan, M. Papaefthymiou, T. Wenisch, M. M. K. Martin, and K. P. Pipe. On-chip Phase Change Heat Sinks Designed for Computational Sprinting. In *Proceedings of the Thirtieth Annual IEEE Semiconductor Thermal Measurement and Management Symposium*, 2014.

[150] S. L. Sheldon and M. R. Guthaus. Package-Chip Co-Design to Increase Flip-Chip C4 Reliability. In *12th International Symposium on Quality Electronic Design (ISQED)*, pages 1–6. IEEE, 2011.

[151] B. Shi, Y. Zhang, and A. Srivastava. Dynamic Thermal Management for Single and Multi-core Processors under Soft Thermal Constraints. In *Proceedings of the 2010 International Symposium on Low Power Electronics and Design*, 2010.

[152] 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.

[153] S. Sivakkumar and A. Pandolfo. Evaluation of Lithium-Ion Capacitors Assembled with Pre-Lithiated Graphite Anode and Activated Carbon Cathode. *Electrochimica Acta*, 65(0):280–287, 2012.

[154] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, June 2003.

[155] N. Soares, J. Costa, A. Gaspar, and P. Santos. Review of Passive PCM Latent Heat Thermal Energy Storage Systems Towards Buildings Energy Efficiency. *Energy and Buildings*, 59: 82–103, Apr 2013.

[156] D. J. Sorin. Fault Tolerant Computer Architecture. *Synthesis Lectures on Computer Architecture*, 2009.

[157] S. Souders. Even Faster Web Sites, 2009. URL `http://stevesouders.com/docs/web20expo-20090402.ppt`.

[158] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. The Case for Lifetime Reliability-Aware Microprocessors. In *Proceedings of the 31st Annual International Symposium on Computer Architecture*, June 2004.

[159] M. D. Stoller, S. Park, Y. Zhu, J. An, and R. S. Ruoff. Graphene-Based Ultracapacitors. *Nano Letters*, 8(10):3498–3502, 2008.

[160] M. A. Suleman, O. Mutlu, J. A. Joao, Khubaib, , and Y. N. Patt. Data Marshaling for Multi-core Systems. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, June 2010.

[161] F. Tan and S. Fok. Thermal Management of Mobile Phone Using Phase Change Material. In *Proceedings of the Ninth Electronics Packaging Technology Conference*, Dec. 2007.

[162] 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.

[163] R. W. Technologies. Llano at Hot Chips, 2011. URL `http://www.realworldtech.com/page.cfm?ArticleID=RWT083111130632`.

[164] 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.

[165] S. S. Tsai, D. M. Chen, V. Chandrasekhar, G. Takacs, N.-M. Cheung, R. Vedantham, R. Grzeszczuk, and B. Girod. Mobile Product Recognition. In *Proceedings of the International Symposium on Multimedia*, 2010.

[166] 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*, pages 159–166, 1989.

[167] R. Velraj, R. V. Seeniraj, B. Hafner, C. Faber, and K. Schwarzer. Heat Transfer Enhancement in a Latent Heat Storage System. *Solar Energy*, 65(3), 1999.

[168] 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.

[169] 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.

[170] G. Venkatesh, J. Sampson, Jack, N. Nathan Goulding-Hotta, S. K. Venkata, M. B. Taylor, and S. Swanson. QsCores: Trading Dark Silicon for Scalable Energy Efficiency with Quasi-specific Cores. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, Nov. 2011.

[171] T. N. Vijaykumar and G. S. Sohi. Task Selection for a Multiscalar Processor. In *Proceedings of the 31st Annual IEEE/ACM International Symposium on Microarchitecture*, Nov. 1998.

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

[173] S. Wang and M. Baldea. Temperature Control and Optimal Energy Management using Latent Energy Storage. *Industrial & Engineering Chemistry Research*, 52(9):3247–3257, 2013.

[174] Y. Wang, Y. Kim, Q. Xie, N. Chang, and M. Pedram. Charge Migration Efficiency Optimization in Hybrid Electrical Energy Storage (HEES) Systems. In *Proceedings of the 2011 International Symposium on Low Power Electronics and Design*, 2011.

[175] M. Ware, K. Rajamani, M. Floyd., B. Brock, J. C. Rubio, F. Rawson., and J. B. Carter. Architecting for Power Management: The IBM POWER7 Approach. In *Proceedings of the 16th Symposium on High-Performance Computer Architecture*, Feb. 2010.

[176] R. Wirtz, N. Zheng, and D. Chandra. Thermal Management Using Dry Phase Change Materials. In *Proceedings of the Fifteenth Annual IEEE Semiconductor Thermal Measurement and Management Symposium*, 1999.

[177] D. A. Wood and M. D. Hill. Cost-Effective Parallel Computing. *IEEE Computer*, pages 69–72, Feb. 1995.

[178] L. Xia, Y. Zhu, J. Yang, J. Ye, and Z. Gu. Implementing a Thermal-Aware Scheduler in Linux Kernel on a Multi-Core Processor. *The Computer Journal*, 53(7):895–903, 2010.

[179] Q. Xie, Y. Wang, M. Pedram, Y. Kim, D. Shin, and N. Chang. Charge Replacement in Hybrid Electrical Energy Storage Systems. In *Proceedings of Asia and South Pacific Design Automation Conference*, Jan 2012.

[180] 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.

[181] 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.