Utilizing Dark Silicon to Save Energy with Computational Sprinting

Computational sprinting activates dark silicon to improve responsiveness by briefly but intensely exceeding a system's sustainable power limit. Sprinting can save energy and improve responsiveness by enabling execution in chip configurations that, although thermally unsustainable, improve energy efficiency. This energy savings can improve throughput even for long-running computations. Repeatedly alternating between sprint and idle modes while maintaining sustainable average power can outperform steady-state computation at the platform's thermal limit.

Arun Raghavan
Laurel Emurian
University of Pennsylvania

Lei Shao
Marios Papaefthymiou
Kevin P. Pipe
Thomas F. Wenisch
University of Michigan

Milo M. K. Martin
University of Pennsylvania

Researchers predict increasingly underutilized chip area (*dark silicon*) with continued CMOS scaling for all designs,[1,2] and the impact of dark silicon in mobile devices is likely to be particularly acute. Because of limited heat-venting capability, researchers project that only 10 percent of transistors on a mobile chip can remain active on a sustained basis.[3] To extract value from dark silicon in such thermally constrained settings, our earlier work proposed *computational sprinting*, an approach to improve responsiveness for interactive applications by briefly exceeding sustainable thermal limits through activating otherwise idle cores and increasing frequency[4] (see the "Computational-Sprinting Overview" sidebar).

Here, we further explore counterintuitive findings regarding the energy implications of sprinting. Our original simulation study naively concluded that sprinting by activating reserve cores would be energy-neutral at best, because it assumed that chip power was solely due to active cores. In fact, real chips incur significant background power overheads (above the idle power) to activate even a single core because of shared "uncore" components, such as caches and interconnects. Sprinting activates dark silicon cores to use these resources more efficiently, and it also lets them idle sooner by completing computation faster. Previous literature has noted this race-to-idle effect.[5-8] However, under thermal constraints, "sprinting to idle" reveals new ways to use dark silicon to conserve energy. By leveraging sprinting to perform a staccato sprint-and-rest execution, wherein the system alternates between

Published by the IEEE Computer Society

Computational-Sprinting Overview

CMOS scaling trends project an inflection point where thermal constraints (especially in mobile devices that employ only passive cooling) preclude sustained operation of all transistors on a chip—a phenomenon called *dark silicon*. However, many mobile applications do not demand sustained performance; rather, they comprise short bursts of computation in response to sporadic user activity. Conventional processors, including their heat sinks, are designed primarily for sustained performance. However, sustained performance isn't the relevant metric for many interactive, mobile applications. Rather, we pose the question: ''What would a system look like if designed to provide responsiveness during bursts rather than with a singular focus on sustained performance?''

Computational sprinting activates otherwise powered-down cores, and boosts voltage and frequency for bursts of intense computation in response to such intermittent usage. During sprinting, chip temperature does not spike instantaneously, although the processor generates heat faster than the system dissipates it. Instead, the system absorbs heat by virtue of its inherent thermal capacitance, which causes temperature to rise over an extended—albeit still short—time interval. When the temperature reaches a threshold value, sprinting terminates (by deactivating reserve cores and reducing frequency), and any remaining computation is completed at the sustainable baseline.

Most materials, such as silicon and metals used in chips, possess thermal capacitance because of their specific heat. For example, the processor package used in this article contains a 20-g internal heat spreader made of copper, which absorbs 188 J of energy to heat up by 25°C—enough to allow a few seconds of sprinting at 50 W even when the platform is cooling-constrained to only dissipate 10 W. Alternatively, a second form of thermal capacitance results from the latent heat of phase change in certain materials. For example, 1 g of wax could absorb 200 J to enable the same amount of sprinting, provided the heat spreads quickly enough to melt all the material.

During sprints, the processor generates heat at a rate that far exceeds the thermal (cooling) and electrical (power delivery and stability) capacities of a typical smartphone-like device. Earlier work, therefore, explored various thermal, electrical, architectural, and software-runtime aspects to effectively facilitate sprinting for short time durations, overcoming the physical challenges inherent in our target environments.[1]

To further validate the feasibility of sprinting, we constructed a simple testbed by constraining the cooling system around an off-the-shelf Intel Sandy Bridge processor so that only one of its four cores, operating at a minimum frequency (1.6 GHz), could be sustained (10 W operating power).[2] Activating all four cores and doubling frequency causes the operating power to increase to 50 W, which is unsustainable. Figure A compares these modes of operation for the Sobel workload. Sprinting speeds up execution by 7× by enabling the task to complete before the temperature reaches the maximum threshold.

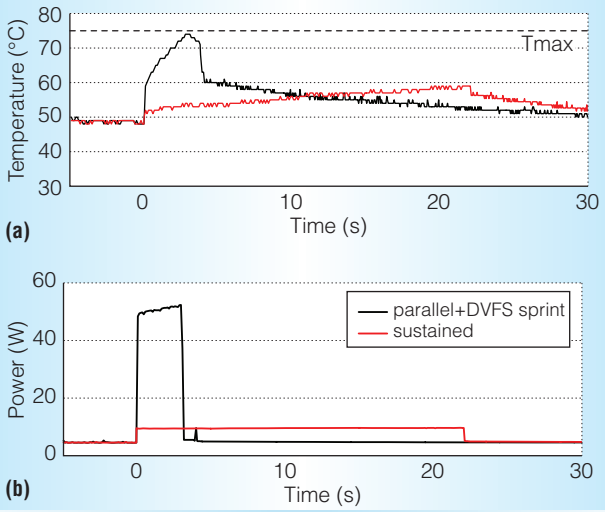

Figure A. Thermal response (a) and input power (b) for sustained `Parallel` sprinting and `Parallel+DVFS` sprinting operation for the Sobel workload. Sprinting speeds up execution by 7×.

References

1. A. Raghavan et al., ''Computational Sprinting,'' *Proc. 18th Symp. High-Performance Computer Architecture* (HPCA 12), IEEE CS, 2012, doi:10.1109/HPCA.2012.6169031.
2. A. Raghavan et al., ''Computational Sprinting on a Hardware/Software Testbed,'' *Proc. 18th Int'l Conf. Architectural Support for Programming Languages and Operating Systems* (ASPLOS 13), ACM, 2013, pp. 155-166.

sprinting and idling at a duty cycle that maintains a thermally sustainable average power, sprinting can use dark silicon to actually improve throughput over conventional steady-state execution at a thermally sustainable pace.

This article develops a simple analytical model to describe the conditions on speedup and platform power under which sprint-to-idle and sprint-and-rest can improve both performance and energy efficiency. We present results from a hardware testbed to empirically validate the model and show that both sprint-to-idle and sprint-and-rest do indeed provide faster and more energy-efficient modes of operation than simple slow-and-steady sustainable execution.

Table 1. Testbed power profile.

Frequency f	Cores N	Total power $P_{total}(N,f)$	Normalized power $P_{total}(N,f)/P_{total}(1,f_{min})$	Peak speedup $S(N,f)$	Mode	$P_{core}(1,f)$	$P_{uncore}(f)$
1.6 GHz	1	~10 W	1×	1×	Sustainable	3.3 W	6.6 W
1.6 GHz	2	~13 W	1.3×	2×	–	3.3 W	6.6 W
1.6 GHz	4	~20 W	2×	4×	Parallel sprint	3.3 W	6.6 W
3.2 GHz	1	~20 W	2×	2×	–	10 W	10 W
3.2 GHz	2	~30 W	3×	4×	–	10 W	10 W
3.2 GHz	4	~50 W	5×	8×	Parallel + DVFS sprint	10 W	10 W

* DVFS: dynamic voltage and frequency scaling.

When does sprinting save energy?

The opportunity to save energy with sprinting arises because of the power required to keep a chip's shared (that is, uncore) components active to support the operation of even a single core. Despite being classified as overhead, this background power is fundamental 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 we use for evaluations, this uncore power is twice the power of a single core at minimum operating frequency (Table 1). System designs embrace this overhead and seek to amortize it by scaling up computation resources (for example, adding more cores) to compute in a more energy-efficient way per operation; a similar argument was made for the cost of parallel-computing systems by Wood and Hill.[9] 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.[5-8]

Although it is desirable to operate in these energy-efficient modes, the thermal constraints that give rise to dark silicon can also preclude such sustained operation. By intermittently activating dark silicon, computational sprinting can reduce the energy per operation via sprint-to-idle. To explain these previously seen advantages,[10] we present an analytical model using the parameters in Table 2. The model separates core (active), uncore (background), and idle power based on empirical data from a real system. The model doesn't explicitly include workload-dependent variation in power consumption, because our results show low variation across the workloads used in the evaluation. For a given frequency, the system we evaluate closely fits a model with a fixed background power and active power growing linearly in direct proportion to the number of active cores (Table 1). 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} (that is, at power $P_{core}(1,f_{min})$) while obtaining a speedup of $S(N,f)$.

Using this model, we analyze the energy impact of sprinting, considering three questions:

- When does sprinting improve energy efficiency per operation?
- When does sprinting result in a net energy savings when also considering the implications of nonnegligible idle power?
- When is repeated sprint-and-rest more energy-efficient than steady computation at sustainable power in thermally constrained environments?

Sprinting to reduce energy per operation

The total energy a system consumes while computing is as follows:

Energy during computation
= (core power + background power)
· computation time

Table 2. Parameters used in energy analysis.

Parameter	Derivation	Meaning
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
$S(N,f)$	$t_{compute}(N,f)/t_{compute}(1,f_{min})$	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: $P_{sustainable} - P_{idle}/P_{total}(N,f) - P_{idle}$	Fraction of total time spent in sprint mode

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

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

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

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

Thus, the relative energy is

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

For more energy-efficient computation (relative energy < 1), the higher-power sprint modes must deliver a minimum speedup:

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

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

$$S(N,f_{min}) > \frac{N + \frac{P_{uncore}(f_{min})}{P_{core}(1,f_{min})}}{1 + \frac{P_{uncore}(f_{min})}{P_{core}(1,f_{min})}} \quad (2)$$

With no background power, we would need ideal, linear speedup ($S(N,f_{min}) = N$) for any additional cores to even be energy-neutral with single-core operation. However, nonzero 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 sublinear speedup can be more energy-efficient than operating in the lowest-power mode. For example, in our evaluation system, in which $P_{uncore}(f_{min})$ is 6.6 W and $P_{core}(1,f_{min})$ is 3.3 W, a speedup exceeding 2× with four cores is sufficient for saving energy.

Implications of idle power

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

$$E_{total}(N,f) = E_{compute}(N,f) + E_{idle}(N,f)$$

$$E_{idle}(N,f) = P_{idle}\left(t_{compute}(1,f_{min}) - \frac{t_{compute}(1,f_{min})}{S(N,f)}\right)$$

$$E_{total}(1,f_{min}) = E_{compute}(1,f_{min})$$

Thus, the relative energy is

$$\text{Relative energy} = \frac{E_{total}(N,f)}{E_{total}(1,f_{min})} = \frac{N \cdot P_{core}(1,f) + P_{uncore}(f) + P_{idle}(S(N,f)-1)}{S(N,f)(P_{core}(1,f_{min}) + P_{uncore}(f_{min}))} \quad (3)$$

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

$$S(N,f_{min}) > \frac{N + \frac{P_{uncore}(f_{min}) - P_{idle}}{P_{core}(1,f_{min})}}{1 + \frac{P_{uncore}(f_{min}) - P_{idle}}{P_{core}(1,f_{min})}} \quad (4)$$

Equation 4 is similar to the previous requirement on speedup (Equation 2), 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 lower than background power in most reasonably engineered systems. In a sprint-enabled system, when sufficient speedup is obtained, it can be possible to use 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 our evaluation system—where P_{idle} is 5 W, $P_{uncore}(f_{min})$ is 6.6 W, and $P_{core}(1,f_{min})$ is 3.3 W, as stated previously—speedup exceeding 3× with four cores is sufficient for saving energy.

Sprint-and-rest

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). However, in a sprint-enabled system, we can also consider an operating regime that alternates between sprint and rest periods. Provided that the sprint periods are short enough to remain within temperature bounds, and that the rest periods are long enough to dissipate the accumulated heat, such a sprint-and-rest operation mode is also sustainable 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 system's thermal power limit 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:

$$P_{sprint\text{-}and\text{-}rest} \le P_{sustainable}$$
$$P_{total}(N,f)\,r_{sprint}(N,f) + P_{idle}(1 - r_{sprint}(N,f)) \le P_{sustainable} \quad (5)$$

Therefore,

$$r_{sprint}(N,f) \le \frac{P_{sustainable} - P_{idle}}{P_{total}(N,f) - P_{idle}}$$

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

$$S_{sprint\text{-}and\text{-}rest}(N,f) = S(N,f)\,r_{sprint}(N,f) \le S(N,f)\,\frac{P_{sustainable} - P_{idle}}{P_{total}(N,f) - P_{idle}}$$

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

$$E_{\text{sprint-and-rest}}(N,f)=\frac{P_{\text{sprint-and-rest}}(N,f)}{S_{\text{sprint-and-rest}}(N,f)}.$$

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

$$\begin{aligned}&\text{Relative energy}\\&=\frac{E_{\text{sprint-and-rest}}(N,f)}{E_{\text{compute}}(1,f_{\text{min}})}\\&=\frac{1}{S(N,f)}\cdot\frac{P_{\text{total}}(N,f)-P_{\text{idle}}}{P_{\text{sustainable}}-P_{\text{idle}}}\end{aligned}\qquad(6)$$

We next evaluate these inferences experimentally.

Methodology: Sprinting testbed design and characterization

To sprint, a chip must offer an operating point where its peak power greatly exceeds the sustainable power dissipation of its cooling system. Existing mobile chips have been designed with peak power envelopes easily dissipated via passive cooling, and thus are inadequate for our study. Instead, we study a sprinting testbed system as a proxy for the thermal characteristics of a future sprint-enabled device. This system uses an Intel Core i7 2600 quad-core Sandy Bridge chip.[10] The chip can operate with one to four cores over a frequency range from 1.6 GHz to 3.2 GHz. Table 1 shows this chip's power and peak performance for the relevant subset of these modes. The power is measured using energy counters that reflect package-level energy consumption.[11] Table 1 shows that, for each frequency, the total power is well approximated as a fixed background power and a per-core power multiplied by the number of active cores.

We reduce the chip's heat-venting capacity by removing its heat sink and tuning its fan to dissipate 10 W, so that the temperature settles at the maximum recommended operating temperature of 75°C when running with a single core at 1.6 GHz; all other modes are not thermally sustainable and hence are sprint modes. The chip idles at ~5 W, causing its initial temperature to settle at 50°C. The chip's internal heat spreader (~20 g of copper) can store up to 188 J of heat for a 25°C temperature increase, allowing several seconds of sprinting with the four-core, 1.6-GHz (`Parallel`) and four-core, 3.2-GHz (`Parallel+DVFS`) modes.

We evaluate the performance and energy impact of sprinting on our test platform using a suite of vision kernels. The kernels' inputs are sized such that each completes within a single sprint without exhausting thermal capacitance; other work investigates cases when thermal capacitance is exhausted midsprint.[10]

Performance and energy impact of sprinting

We first measure the speedup provided by sprinting. We then predict available energy savings from sprinting-to-idle according to the model and compare those predictions to empirical energy measurements.

Speedup

When sprinting is employed with four cores at 1.6 GHz, the maximum potential speedup over the single-core baseline is 4×; with `Parallel+DVFS` sprinting, maximum speedup is 8× (4× cores, 2× frequency). Figure 1a shows the achieved speedups for our vision kernels: `Parallel+DVFS` enables 6.3× speedup on average, whereas `Parallel` sprinting achieves 3.5× speedup on average. These speedups imply that sprinting allows this system to complete in just a few seconds what would have taken more than 15 seconds if constrained to operate only in sustainable (nonsprinting) mode.

Energy efficiency of sprinting

We can predict the energy impact of sprinting from the measured speedups and the models developed earlier. Relative to the sustainable baseline ($P_{\text{core}}(1,f_{\text{min}})$), Equation 1 with $S(N,f) = 6.3\times$ predicts the active energy to be 0.79× the baseline for `Parallel+DVFS`, and 0.57× for `Parallel+DVFS` sprinting, with $S(N,f) = 3.5\times$ (substantial energy savings). The lower component (darker) of Figure 1b shows the experimentally observed energy for the duration of the sprint. The measured energy (0.77× and 0.60×) for both sprinting

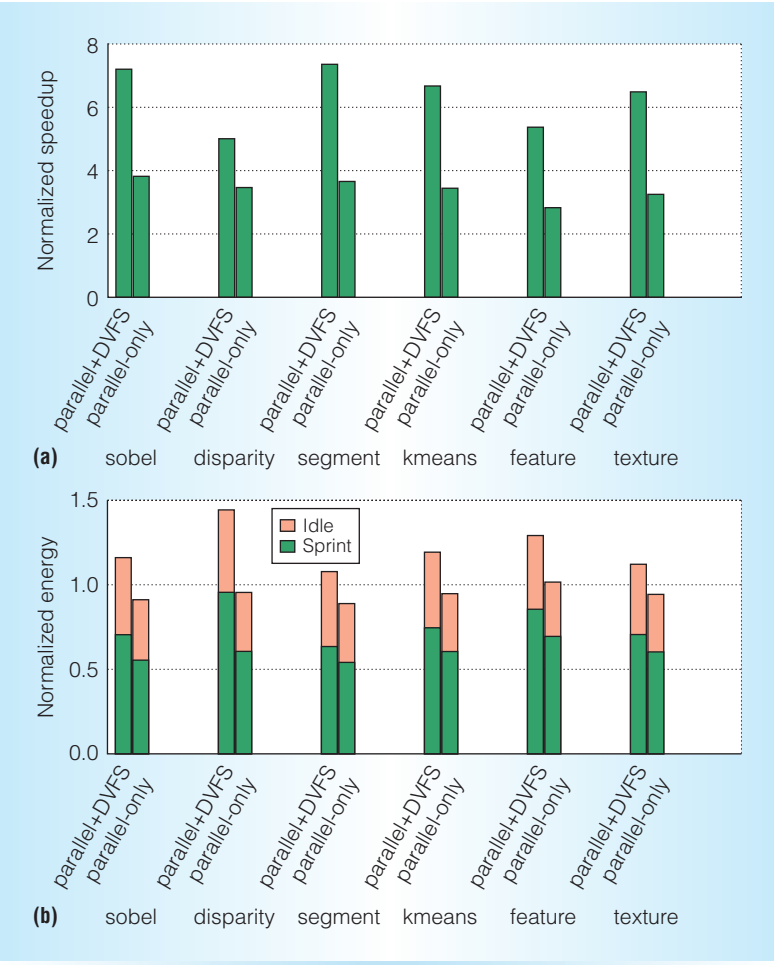

Figure 1. Speedup ($S(N,f)$) (a) and energy breakdown ($E_{compute}(N,f)$, $E_{idle}(N,f)$) normalized to the one-core 1.6-GHz sustainable baseline ($E_{compute}(1,f_{min})$) (b) for four cores at 3.2 GHz and 1.6 GHz. Sprinting with parallelism and DVFS results in an average speedup of 6.3× (a) using 23 percent less energy to perform the computation (dark component in part b). However, idle energy after sprinting (light component in part b) causes a total energy loss of 20 percent. Sprinting with parallelism alone results in lower average speedup (3×), but saves energy (6 percent), even considering idle energy.

modes closely matches the model prediction and confirms that sprinting enables lower energy per operation.

Energy efficiency of sprint-to-idle

After a computation completes during a sprint, however, the system continues to consume some energy while idle. The model accounts for this idle energy in Equation 3. Relative to sustained operation, the predicted total energy is 1.20× the baseline for `Parallel+DVFS` sprinting (a net energy loss), whereas `Parallel` sprinting consumes 0.93× the baseline energy (continuing to provide a net energy savings). We account for the additional idle energy in the upper, lighter component of each bar in Figure 1b. Again, the measured energy confirms the model (21 percent energy overhead with `Parallel+DVFS` sprinting, and 6 percent energy savings with `Parallel` sprinting).

Sprint-and-rest for long running computations

We experiment with sprint-and-rest in our system with both the `Parallel` and `Parallel+DVFS` modes of sprinting. From Equation 5, for the `Parallel` sprint drawing 20 W, the fraction of time spent in sprint mode cannot exceed 1/3.1 ($r_{sprint}(N,f) = 1:3.1$) to provide a sustainable average power. To avoid overheating during an individual sprint, sprint duration for `Parallel` sprinting cannot exceed 20 seconds. Thus, we selected a sprint duration of 5 seconds and a rest duration of 10.5 seconds ($r_{sprint}(N,f) = 1:3.1$). Similarly, for `Parallel+DVFS` sprinting at 50 W, we selected the sprint duration as 1.5 seconds (less than the 3 seconds maximum sprint duration), and a rest duration of 12.3 seconds ($r_{sprint}(N,f) = 1:9.1$). Substituting these values in Equation 6, we would expect `Parallel` sprinting to be 23 percent more energy-efficient, and `Parallel+DVFS` sprinting to be 22 percent less energy-efficient, compared to sustained execution at a constant 10 W of power.

Figure 2a shows the power traces for the Sobel workload executed on the testbed for more than 8 minutes with sustained and sprint-and-rest modes (for both `Parallel` and `Parallel+DVFS` sprinting) under the previously discussed duty cycles. Figure 2b compares the resulting cumulative work done when operating in these modes. The `Parallel+DVFS` sprint-and-rest mode underperforms sustained execution at the sustainable thermal limit by 21 percent. However, the `Parallel` mode of sprint-and-rest performs 20 percent more work over sustained operation on average.

The above results provide ample motivation for chip designers to further optimize idle power; although the chip used

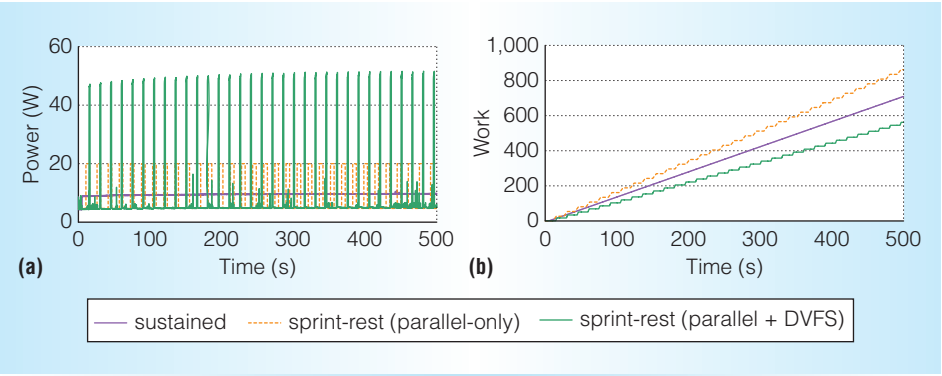

Figure 2. Comparison of power (a) and cumulative work (b) done with sprint-and-rest and sustained computation. Although operating with the same average power, repeated sprint-and-rest using parallelism alone computes faster than steady, sustained operation in the example system. Increasing frequency (Parallel+DVFS) is less energy efficient and underperforms sustained computation for the same average power.

for the evaluation already achieves 10-to-1 ratios between peak and idle power, the analytical as well as empirical results indicate that energy efficiency gains of sprinting would increase if idle power is further reduced. Thus, by keeping dark silicon as dark as possible when idle, and operating dark silicon beyond sustainable power when active, computational sprinting has the potential to continue harnessing Moore's law to deliver intense performance when necessary, and even save energy in the process. MICRO

References

1. H. Esmaeilzadeh et al., ''Dark Silicon and the End of Multicore Scaling,'' *Proc. 38th Ann. Int'l Symp. Computer Architecture* (ISCA 11), ACM, 2011, pp. 365-376.
2. M.B. Taylor, ''Is Dark Silicon Useful? Harnessing the Four Horsemen of the Coming Dark Silicon Apocalypse,'' *Proc. 49th Design Automation Conf.,* IEEE CS, 2012, pp. 1131-1136.
3. R. Merritt, ''ARM CTO: Power Surge Could Create 'Dark Silicon,' '' *EE Times,* 22 Oct. 2009; http://www.eetimes.com/electronics-news/4085396/ARM-CTO-power-surge-could-create-dark-silicon.
4. A. Raghavan et al., ''Computational Sprinting,'' *Proc. 18th Symp. High-Performance Computer Architecture* (HPCA 12), IEEE CS, 2012, doi:10.1109/HPCA.2012.6169031.
5. S. Albers and A. Antoniadis, ''Race to Idle: New Algorithms for Speed Scaling with a Sleep State,'' *Proc. 23rd Ann. ACM-SIAM Symp. Discrete Algorithms* (SODA 12), SIAM, 2012, pp. 1266-1285.
6. X. Li et al., ''Performance Directed Energy Management for Main Memory and Disks,'' *Proc. 11th Int'l Conf. Architectural Support for Programming Languages and Operating Systems,* ACM, 2004, pp. 271-283.
7. D. Meisner, B.T. Gold, and T.F. Wenisch, ''PowerNap: Eliminating Server Idle Power,'' *Proc. 14th Int'l Conf. Architectural Support for Programming Languages and Operating Systems,* ACM, 2009, pp. 205-216.
8. A. Miyoshi et al., ''Critical Power Slope: Understanding the Runtime Effects of Frequency Scaling.'' *Proc. 16th Int'l Conf. Supercomputing,* ACM, 2002, pp. 35-44.
9. D.A. Wood and M.D. Hill, ''Cost-Effective Parallel Computing,'' *Computer,* Feb. 1995, pp. 69-72.
10. A. Raghavan et al., ''Computational Sprinting on a Hardware/Software Testbed,'' *Proc. 18th Int'l Conf. Architectural Support for Programming Languages and Operating Systems* (ASPLOS 13), ACM, 2013, pp. 155-166.
11. *Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2,* Aug. 2012.

Arun Raghavan is a PhD candidate in the Department of Computer and Information Science at the University of Pennsylvania.

His research interests include parallel-computer architectures and programming models. Raghavan has a BEng in electronics and communications engineering from R.V. College of Engineering, India.

Laurel Emurian is a PhD candidate in the Department of Computer and Information Science at the University of Pennsylvania. Her research interests include energy-efficient systems and mobile architecture. Emurian has an MSE in computer and information science from the University of Pennsylvania.

Lei Shao is a PhD candidate in the Department of Mechanical Engineering at the University of Michigan. His research interests include microscale heat transfer and energy conversion. Shao has an MS in mechanical engineering from the University of Michigan.

Marios Papaefthymiou is a professor in the Department of Electrical Engineering and Computer Science and Chair of Computer Science and Engineering at the University of Michigan. He is also a cofounder and chief scientist of Cyclos Semiconductor, a start-up company specializing in energy-efficient chips for power-critical applications. Papaefthymiou has a PhD in electrical engineering and computer science from the Massachusetts Institute of Technology.

Kevin P. Pipe is an associate professor in the Department of Mechanical Engineering and holds joint appointments with the Applied Physics Program and Electrical Engineering and Computer Science Department at the University of Michigan. His research interests include microscale heat transfer, optoelectronic devices, thermoelectric energy conversion, scanning-probe techniques, photovoltaic energy conversion, and organic and hybrid organic and inorganic devices. Pipe has a PhD in electrical engineering from the Massachusetts Institute of Technology.

Thomas F. Wenisch is the Morris Wellman Faculty Development Assistant Professor of Electrical Engineering and Computer Science at the University of Michigan and a member of the Advanced Computer Architecture Lab. His research focuses on computer architecture with emphasis on multiprocessor and multicore systems, multicore programmability, smartphone architecture, datacenter architecture, and performance evaluation methodology. Wenisch has a PhD in electrical and computer engineering from Carnegie Mellon University.

Milo M. K. Martin is an associate professor in the Department of Computer and Information Science at the University of Pennsylvania. He coleads Penn's Computer Architecture and Compilers Group. His research interests include multiprocessor and multicore computer architecture, compiler and hardware support for security, and programming models for next-generation architectures. Martin has a PhD in computer science from the University of Wisconsin-Madison.

Direct questions and comments about this article to Arun Raghavan, LVN 302, 3330 Walnut St., Philadelphia, PA 19104; arraghav@cis.upenn.edu.