Adaptive Sprinting: How to Get the Most Out of Phase Change Based Passive Cooling

Fulya Kaplan and Ayse K. Coskun
ECE Department, Boston University, Boston, MA – {fkaplan3, acoskun}@bu.edu

***Abstract*—CMOS scaling trends lead to elevated on-chip temperatures, which substantially limit the performance of today's processors. To improve thermal efficiency, Phase Change Materials (PCMs) have recently been used as passive cooling solutions. PCMs store large amount of heat at near-constant temperature during phase change, allowing strategies such as *computational sprinting*. While existing sprinting methods allow short performance boosts, there is significant unexplored potential in improving performance on systems with PCM-enhanced cooling. To this end, this paper proposes a novel runtime management policy driven by observations that are not captured by prior techniques: (i) PCM melts non-uniformly due to spatially heterogeneous on-chip heat distribution; (ii) power consumption during sprinting is highly application dependent and assuming a fixed sprinting power leads to lower thermal efficiency; (iii) if we monitor the remaining PCM energy at various locations, we can utilize the PCM heat storage capability much more efficiently. The proposed Adaptive Sprinting policy exploits these observations to extend sprinting duration for increased performance gains. Our policy monitors the remaining PCM energy corresponding to each core at runtime, and using this information, it decides on the number, the location and the voltage-frequency (V/f) setting of the sprinting cores. Experimental evaluation including a detailed phase change thermal model demonstrates 29% performance improvement, 22% energy savings, and 43% energy delay product (EDP) reduction on average, compared to prior strategies.**

I. Introduction

Technology scaling accompanied with saturation in voltage scaling has led to an increase in on-chip power density at a rate that exceeds the heat removal ability of current cooling systems. This increase in power density introduces the phenomenon referred to as *dark silicon*; i.e., not all the transistors on the chip can be powered on at full performance without exceeding the *thermal design power* (TDP). TDP is the maximum amount of power that the chip can dissipate while operating below the critical temperature threshold in the *sustained mode*. ITRS roadmap projections estimate more than 50% dark silicon area for the 8nm technology node [1].

Recent research explores runtime mechanisms to efficiently utilize the chip area to maximize performance under thermal constraints in the dark silicon era [2][3][4]. Intel's Turbo Boost [2] and AMD's Turbo CORE [3] exploit the temperature headroom to operate the cores at higher V/f settings for short amounts of time. *Computational sprinting* exceeds the TDP by activating all of the dark silicon cores during short bursts of high intensity computing demands [4].

On the cooling side, the use of PCMs has been investigated as a passive cooling technique [4][5][6][7]. PCMs are compounds that store large amounts of *latent heat* during phase change from solid to liquid. PCM absorbs this heat at a near-constant temperature and hence acts like a large thermal capacitor. These properties have led to the use of PCM in cooperation with *computational sprinting* [4][5][6].

The current research focuses on using PCM in the context of *computational sprinting* to extend sprinting duration. Prior techniques on *computational sprinting* alternate between sprinting with all cores and not sprinting by switching to idle mode or single core operation [4][5][6][7]. However, existing techniques ignore the following observation: due to the inherent heterogeneous heat distribution across a chip, different parts of PCM melt at different rates depending on their location. Thus, this type of sprinting with an *all or nothing* approach wastes the yet unused PCM capacity, leading to substantially suboptimal performance. Similarly, existing techniques do not consider the application's power consumption and assume a fixed sprinting power. These policies operate under the worst case power consumption scenario, and thus, potentially incur performance losses for lower power applications. In fact, factors such as application's power consumption or the number of cores to sprint with are significant factors in determining the sprinting duration. Sprinting policies that do not consider these factors cannot exploit full benefits of PCM.

For example, prior techniques [4][7] start running applications with 16 threads and in case of thermal violation, they switch to either idle mode or single thread operation, assuming all cores exhaust their PCM capacity at the same time. However, our work shows that there is opportunity to continue sprinting with a lower number of threads (instead of a single thread) using the cores that still have remaining unmelted PCM on them. In this way, we can provide performance benefits over the existing policies.

This paper addresses this performance gap by introducing a novel PCM-aware Adaptive Sprinting policy that decides on the number of sprinting cores, their locations and their V/f settings at runtime. Our policy is application-aware and it does not rely on a priori assumptions about the application's power consumption. Instead, it takes actions by tracking the PCM capacity. Our main contributions are as follows:

- We show that for a system with PCM, melting occurs heterogeneously at different parts of the PCM (Section IV). Thus, even if some cores reach their thermal threshold, there is an opportunity to continue sprinting with other cooler cores that have not yet exhausted their PCM capacity.
- We propose a PCM-aware Adaptive Sprinting policy to improve the performance of multithreaded workloads on systems with PCM (Section IV). Our policy extends sprinting duration by allowing sprinting with a lower/higher number of cores based on the remaining PCM capacity and by choosing an efficient V/f setting.

- Using a detailed PCM thermal model (Section III) we show that our policy provides **29%** higher performance, saves energy by **22%**, and reduces EDP by **43%** on average compared to the best performing sprinting strategy (Sec. V).

II. Related Work

Thermal management of computing systems with PCM has been receiving attention. Recent studies explore the benefits of using PCM as a heat spreader or heat sink enhancer [8][9][10][11][12]. Tan et al. use computational fluid dynamics (CFD) simulations for thermal analysis of a mobile phone with a PCM filled heat storage unit [9]. Alawadhi et al. study the effectiveness of a thermal control unit composed of PCM and a thermal conductivity enhancer on a portable electronic device [8]. Other research focuses on designing hybrid heat sinks that use air-cooling and PCM together for energy savings [10][11]. Low conductivity of PCM significantly limits its potential benefits. Using metal matrix-PCM composites as heat spreaders in mobile electronic devices addresses this issue [12].

Other work in the area proposes using PCM in cooperation with the performance boosting policies to increase system efficiency [4][5][6][7]. Raghavan et al. introduce *computational sprinting*, which is temporarily exceeding the TDP of the chip to improve the responsiveness during short bursts of computation [4]. In their proposed sprinting technique, all of the cores are activated at the highest V/f setting until the cores hit a temperature threshold, after which the execution continues with a single core. Their follow-up work demonstrates the feasibility of sprinting on a hardware/software testbed [5]. They also introduce the concept of *sprint pacing*, where the cores sprint at a lower frequency when half of the thermal capacity is consumed. Other research develops techniques to sprint periodically for longer durations [6][7]. *Safe computational re-sprinting* policy targets periodic tasks with hard deadlines and finds the minimum required PCM latent heat capacity to guarantee re-sprinting at full power [6]. Shao et al. consider repeated sprints with a fixed duty cycle, which is the ratio of the sustained power over sprint power [7]. They implement their technique on a thermal test chip with an on-chip phase change heat sink as a proxy for a smart phone processor.

Our work is the first to consider the heterogeneous melting of the PCM and exploit this heterogeneity to extend sprinting duration. Our policy dynamically decides on the number, location and the V/f setting of the sprinting cores, thus provides continued sprinting with the cores that have available PCM capacity. Moreover, our policy does not depend on assumptions such as consuming a fixed power value during sprinting. In this way, it provides performance benefits regardless of the application power consumption.

III. Methodology

A. Phase Change Thermal Model

We use the detailed phase change model proposed in our prior work [13], which was validated against a CFD model and provided 0.27^oC RMS error. This model was integrated into a compact thermal simulator, HotSpot [14], which considers both vertical and lateral heat flow. The PCM model uses the basic 3D stacking feature in HotSpot, allowing the user to define multiple layers of any desired material. Grid-level simulation

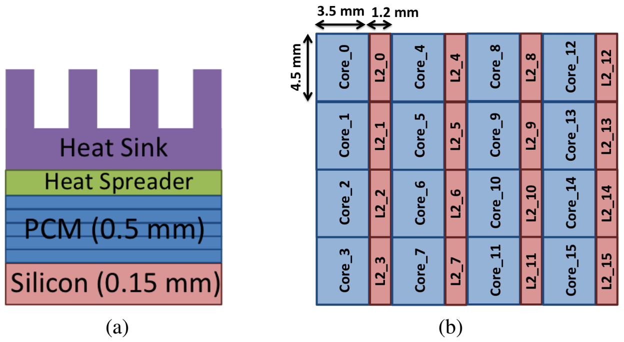


Fig. 1: (a) Chip package incorporating PCM; (b) Layout of our target system.

granularity provides fine-grained simulation by dividing the floorplan into small cells and computing the temperature for each grid cell.

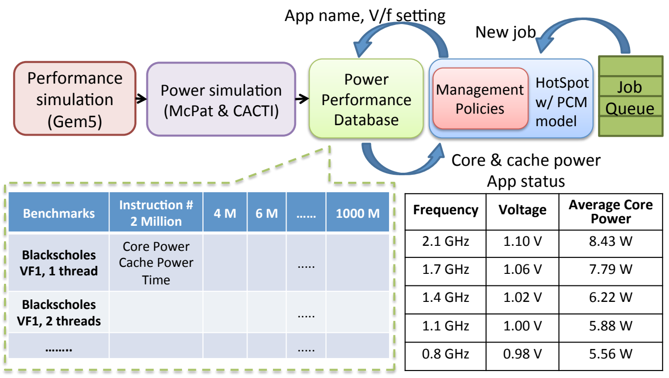
This model we adopt [13] assigns a nonlinear temperature-dependent specific heat capacity to the PCM to model phase change behavior. Phase change from solid to liquid occurs over a temperature interval, during which time the specific heat capacity of the PCM is set to a very high value compared to the values in solid and liquid phases. A very high specific heat capacity during phase transition indicates a very low rate of change of temperature, thus mimics the close-to-constant temperature behavior during melting. Integral of the specific heat capacity over the temperature interval represents the *latent heat of fusion* stored in the PCM.

To implement the model in HotSpot, we insert a layer of PCM that lies on top of the silicon layer as shown in Figure 1(a). The PCM layer has the same layout as the block-level silicon layer layout but it does not dissipate any power. We modify HotSpot to define the melting point and latent heat of fusion of the PCM. Each PCM grid cell is assigned a temperature-dependent specific heat capacity as follows:

$$C_{p,pcm}(T) = \begin{cases} c_{ps} = c_{pl} & T < T_1 \quad or \quad T > T_2 \\ c_{tr} & T_1 \leq T \leq T_2 \end{cases} \tag{1}$$

where c_{ps}, c_{pl}, and c_{tr} are the specific heat capacities of solid, liquid, and phase transition states, respectively. T_1 is the onset temperature and T_2 is the end temperature of the phase transition. In our experiments, we use a transition temperature interval of 1^oC and melting temperature is the center point of the (T_1, T_2) interval. We set c_{ps} = c_{pl} = $1.57 \cdot 10^6 J/m^3K$, and c_{tr} = $244.3 \cdot 10^6 J/m^3K$, which correspond to *cerrobend* PCM. The specific heat capacity of each PCM grid cell is updated based on its temperature at every time step of the simulation following Equation (1). In this way, the model captures the transient temperature behavior of the PCM for a given power trace. The melting pace of the PCM varies depending on the core/chip power consumption. The PCM model responds to changes in the power consumption (due to dynamic voltage frequency scaling (DVFS), etc.) at runtime by computing the temperature of each grid cell based on its corresponding specific heat capacity and the current power consumption at each time step. By using the grid cell granularity, the model also accounts for heterogeneous melting of the PCM (i.e., PCM portions that are located above the hotter parts of the chip melt while the other parts remain in the solid phase).

For the PCM layer, we assume a highly thermally conduc-



Benchmarks	Instruction # 2 Million	4 M	6 M	......	1000 M
Blackscholes VF1, 1 thread	Core Power Cache Power Time			.....	
Blackscholes VF1, 2 threads				.....	
....				.....	

Frequency	Voltage	Average Core Power
2.1 GHz	1.10 V	8.43 W
1.7 GHz	1.06 V	7.79 W
1.4 GHz	1.02 V	6.22 W
1.1 GHz	1.00 V	5.88 W
0.8 GHz	0.98 V	5.56 W

Fig. 2: Performance, power and temperature simulation framework.

tive PCM-copper matrix as suggested in recent work [7]. We use a PCM fraction of 80%, which provides effective thermal conductivity of $75.4W/mK$ with a small sacrifice of the latent heat capacity [7]. We set the PCM thickness as 0.5mm and the melting point as 75^oC.

B. Full System Simulation

We simulate a 16-core processor with private L2 caches as shown in Figure 1(b). The core architecture is based on the AMD Opteron 6172 processor manufactured using a 45 nm SOI process. The architectural parameters for the cores and caches are taken from recent work [15].

Our simulation framework consists of microarchitectural performance simulation (Gem5 [16]), power simulation (McPat [17] and CACTI [18]), temperature simulation (HotSpot), and a database that decouples time-consuming performance and power simulation from thermal simulation (See Figure 2).

We run each benchmark for 1 billion instructions in detailed mode in their parallel phase, and collect performance statistics every 2 million instructions with a total of 500 samples. We calibrate the McPat dynamic core power values based on real measurements collected on the AMD Opteron 6172 processor. We scale the CACTI values based on cache access rates. Figure 2 shows the available V/f levels and the corresponding average core powers for our processor. We use the HotSpot default package properties, except that we use a 1mm thick heat sink with 0.2 K/W convection resistance to represent a system without an advanced heat sink.

The database maintains power and time information for each 2 million instruction frame for each application at all possible thread counts and V/f settings. At every sampling interval, HotSpot polls the database for acquiring the power data of the corresponding benchmark at a specific instruction count. As each cell in the database represents an instruction frame and not time, we can switch from one V/f setting to another by reading from the cell of the desired V/f setting in the next instruction frame. Thus, we can apply DVFS policies or change the thread count (i.e., thread packing) at runtime. This framework has acceptable accuracy because (i) each core has private caches, (ii) V/f scaling is applied to all cores at the same time (which is reasonable for the type of multithreaded benchmarks we run), and (iii) thread packing overhead is small (see Section IV-A). For evaluating any policy that uses DVFS or thread migration, we include the DVFS and migration

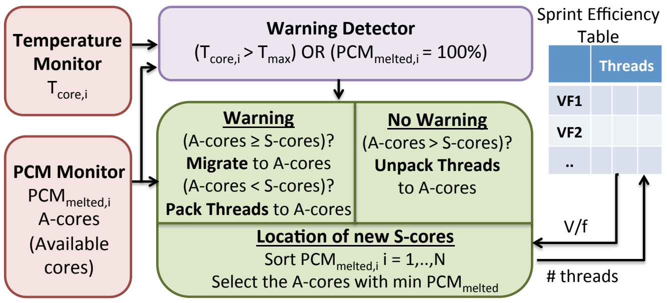


Fig. 3: Proposed adaptive sprinting policy flowchart.

overheads, which are reported as less than 200us [19] and 1ms [20], respectively.

We run benchmarks from the PARSEC Suite [21] as our workload. For each benchmark, we generate performance and power traces at various thread counts (i.e., 1, 2, 4, 6, 8, 10, 12, 14, and 16) using the sim-large input set. We assume equal power consumption for individual threads of an application as inter-thread differences are minimal for PARSEC running on the Opteron CPU.

IV. Management Policies

A. Proposed PCM-Aware Adaptive Sprinting

Based on our experiments using the validated detailed PCM thermal model, we observe that even if all cores consume the same power, they use the PCM capacity at different rates. Individual threads of an application in our experiments consume equal power, thus, the thermal variation is due to the location of the cores. For example, the center cores typically get hotter and force the center part of the PCM to melt faster. When center cores exhaust their PCM capacity and hit a temperature threshold, the side cores still have thermal headroom to continue sprinting. Existing sprinting policies [4][5], however, assume that the cores use up the PCM capacity equally over time, thus, if there is a thermal violation, they either switch to a single core operation or put all cores to idle state.

The **goal** of our adaptive sprinting policy is to operate in sprinting mode as long as possible by leveraging this observation and exploiting the PCM capacity to near exhaustion. By monitoring the PCM state, our policy determines how much sprinting capability is left for each core. We also determine the most sprint-efficient V/f setting using a lookup table. Based on this information, the policy decides on (a) the number, (b) the locations, and (c) the V/f setting of the sprinting cores. The policy changes the number of sprinting cores at runtime by thread packing (or unpacking), which is pinning the threads to a lower (or higher) number of cores [5][22][23].

Figure 3 gives an overview of our adaptive sprinting policy. We first introduce the terminology we use to describe our policy. The active cores are the *sprinting cores* (*S-cores*). *Available cores* (*A-cores*) are the cores which have used less than 100% of their PCM capacity, and have lower temperature than the critical temperature (i.e., T_{max} = 80^oC). *A-cores* can be active or idle at a given time. A *warning* is raised if for any of the cores, the PCM portion that lies above that core is 100% melted **or** if the core temperature exceeds T_{max}.

Our policy checks for a *warning* every 50 ms (a temperature sampling rate that incurs negligible overhead in real systems) and if there is a *warning*, it determines the number of *A-cores* by checking their PCM capacity. If the number of *A-cores* is higher than or equal to the number of *S-cores*, we merely migrate the threads to the *A-cores*. If not, we continue sprinting with fewer cores by packing the threads to the *A-cores* (i.e., binding the threads to a lower number of cores). While thread packing, to determine the location of the new *S-cores*, we sort the used PCM capacity of the cores and select the ones that have exhausted the least amount of PCM capacity. If we are left with no *A-cores*, we put all cores to idle state until some portion of the PCM capacity is recovered (i.e., 10%).

In order to determine the V/f setting at a given time, we follow an offline analysis approach. For this purpose, we run all benchmarks for each of the {thread count, V/f setting} pair when no management policy is applied. We record the original application running times (T_{run}) and the number of instructions executed ($Inst_{spr}$) until the first thermal violation. T_{run} is a measure of performance for the given pair. $Inst_{spr}$ represents how much work can be done when sprinting at a given setting until thermal violation. Based on these recordings, we define a new metric, *sprint efficiency* = $Inst_{spr}$ / T_{run}. Choosing the pair with higher *sprint efficiency* corresponds to choosing a configuration with higher performance while considering the tradeoff between power and allowed sprinting capability. We create a lookup table of *sprint efficiency* values and our policy selects the V/f setting with the maximum *sprint efficiency* for a given thread count.

Our policy also addresses the fact that while the *S-cores* are using up the PCM capacity in parts of the chip, PCM capacity is being recovered around the idle cores. For example, PCM recovery may occur when a benchmark enters a low power phase. Thus, in case of no *warning*, the policy checks if there are more *A-cores* than the current number of *S-cores*. If there are, sprinting continues with the number of *A-cores* by *unpacking* the threads.

Monitoring the PCM capacity: An important aspect of our policy is that it takes actions based on the current PCM state. We monitor the percentage of melted PCM for each core individually (i.e., for each core, we track the latent heat stored in the PCM portion that lies on top of that core and has the same area as that core). In this way, at a given time, we know the sprinting capability for each core. In our HotSpot simulations, we monitor the percentage of melted PCM corresponding to each core using Equation (1) (i.e., % of melted PCM for a grid cell increases linearly within the (T_1, T_2) interval).

In a real-life implementation, such estimations can be done by using temperature measurements and the thermal resistance values as follows: For each core, we find the heat entering the PCM from the silicon layer and exiting the PCM to the air. Assuming we have thermal sensors on the cores and on the corresponding locations of the PCM layer, we can estimate the net heat entering the PCM by using the formula [6]:

$$P_{NET,i} = \frac{T_i - T_{PCM,i}}{R_{Si\_to\_PCM}} - \frac{T_{PCM,i} - T_{AMB}}{R_{PCM\_to\_AIR}} \quad (2)$$

$$E_{STORED,i,(t)} = E_{STORED,i,(t-1)} + P_{NET,i} \times t_{sampling} \quad (3)$$

where T_i, $T_{PCM,i}$ and T_{AMB} are the temperatures of core i, the PCM unit corresponding to core i, and ambient air, respectively. $R_{Si\_to\_PCM}$ and $R_{PCM\_to\_AIR}$ are the equivalent thermal resistances from silicon to PCM and from PCM to air. $E_{STORED,i,(t)}$ is the latent heat energy stored in the PCM unit i at time t and $t_{sampling}$ is the sampling interval. In order to fully melt, PCM has to store an amount of energy that is equal to its latent heat of fusion. Equation (3) accumulates this energy during melting. Most current processors have a temperature sensor per core. For larger systems where temperature sensors for each core and each PCM unit are unavailable, temperature estimation techniques can be applied [24].

PCM Monitor Implementation on a Testbed: We demonstrate the applicability of a PCM monitor on an in-house hardware testbed with a PCM container on top. We measure the average core and PCM temperatures using the on-chip temperature sensors and a thermocouple, respectively. We implement the soft PCM sensor described by Equations (2-3) assuming a single PCM unit. We carry out experiments at various V/f settings, applications, and observe consistent behavior (i.e., the reported stored energy rises from zero to the latent heat of the PCM over time with a slope that is consistent with the power consumption). We measure the PCM monitor overhead (including the temperature sensing and the calculations) in terms of CPU utilization on our testbed, which is less than 0.4% when $t_{sampling} = 50ms$.

Performance Impact of Thread Packing: Our policy decreases the number of sprinting cores by binding the threads to a subset of available cores. In that case, the active threads get multiplexed on the active cores, which may incur performance penalty due to synchronization and context switches. For example, binding all 16 threads to a single core may give worse performance than running with a single thread. For PARSEC benchmarks, prior work reports that the performance degradation due to thread packing is less than 3.6% (for an 8-core system [23]) and is 7.3% on average (when packing 12 threads to 4 cores [22]). For benchmarks whose performance is severely affected by thread packing, a task-queue based worker thread execution framework can mitigate this effect [5]. In our simulations using the *adaptive sprinting* policy, we observe that the number of sprinting cores does not fall below 10. Thus, we assume that the performance of packing the threads to N cores is equal to running with N threads with negligible additional penalty.

B. Baseline Policies

Truncated Sprints: This policy [4] activates all cores of a system at the highest V/f level (i.e., full intensity) during sprinting. Sprinting is truncated if the PCM capacity is fully exhausted or if any core temperature exceeds T_{max}. Upon sprint truncation, execution continues on a single core and all other cores are put to idle mode until the application finishes (i.e., *sustained mode*). For applications with running times shorter than the available sprinting duration, *truncated sprints* work well. However, as the application running time gets longer, more time is spent in the *sustained mode*, which overshadows the benefits of sprinting. We implement an improved version of this policy and use it for comparison in Section V. After a truncated sprint, we allow re-sprinting if some portion of the PCM capacity is recovered (i.e., 10% per core), as opposed to running in *sustained mode* until the benchmark execution ends.

Fixed Duty Cycle Sprinting: This policy has been proposed for extended computations [5][7]. It alternates between the sprint and rest modes (i.e., all cores are put to idle state) based on a duty cycle. Duty cycle (D) is determined by the ratio between the sustained power and the sprint power to allow enough time to cool down after sprinting. For example, for 1W of sustained power and 10W of sprint power, $D = 1 : 10$. Assuming a sprinting duration (i.e., the time it takes to reach T_{max} while sprinting) of 1s, this corresponds to 9s of rest time.

Some limitations of this policy are as follows: (1) It assumes a fixed sprinting power and D for all benchmarks. However, power consumption during sprinting is workload-dependent. Having a fixed duty cycle requires considering the worst case scenario (i.e., the highest possible sprinting power) while setting D in order to avoid thermal violations. Thus, for benchmarks that consume lower power, rest time is longer than needed, which incurs performance penalties. (2) It poses significant performance loss in some cases. For example, an application that originally completes in a little over 1s would wait for 9s in rest mode to complete the remaining small portion of the work. We experimentally determine the $D = 1 : 3.92$ and the sprinting duration (561ms) for our system based on the idle and highest sprinting power ($2.63W : 10.3W$).

Sprint Pacing: Prior work [5] proposes this policy to account for a larger range of application lengths. This policy sprints with full intensity until half of the PCM thermal capacity is consumed. After that, it switches to a lower intensity sprint by keeping all cores active, but changing to the lowest V/f setting. However, prior work does not address how this policy behaves in case of a thermal violation at the lowest V/f setting. Thus, we implement two different versions of this policy: *sprint pacing*, which does not take any action after switching to the lower intensity sprinting, and *modified sprint pacing*, which puts the cores to idle mode (until 10% of the PCM capacity is recovered) if a thermal violation occurs during the lower intensity sprint.

Reactive DVFS: We implement a *reactive DVFS* policy, which represents the DVFS policies in current processors, as a baseline that is oblivious to the PCM state. This policy decreases the V/f setting by one step upon temperature violation in any of the cores. If the violation occurs at the lowest V/f setting, all active cores are put to idle mode. After cooling down and recovering a certain thermal headroom (i.e., 2^oC) to T_{max}, cores continue executing and V/f setting is increased in steps.

V. Results

The following set of results compares the performance impact of sprinting policies. As our original power and performance database corresponds to very short running times (less than 300ms) that are lower than the PCM melting time, we loop each application 50 times. We compare our proposed adaptive sprinting policy against the baseline policies (Section IV-B) and the case where no management policy is applied. In the no management case, benchmarks run using 16 threads at the highest V/f setting, which gives the ideal performance.

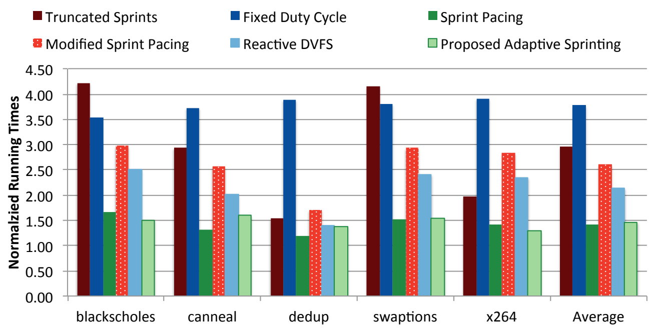


Fig. 4: Running times of the benchmarks normalized to the no management (ideal) case for each application.

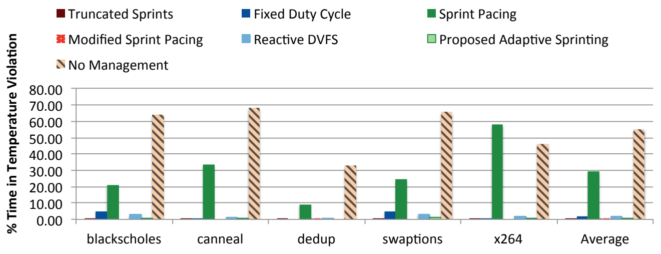


Fig. 5: Percentage of thermal violation (time spent above T_{max}).

Figure 4 shows the running times of the individual benchmarks normalized to the no management case. As indicated, *truncated sprints* and *fixed duty cycle* policies result in the worst performance. Performance of benchmarks such as *blackscholes* and *swaptions* are severely degraded by *truncated sprints* (up to 4.2x of their ideal value). This is because their performance scale well with the number of threads, thus, truncating a sprint leads to losing the benefit of performance scalability. *x264* scales well too, but its performance is not as severely affected by *truncated sprints*. This is because *x264* consumes lower power than the other two benchmarks, thus, it spends more of its time in the sprinting mode.

Fixed duty cycle sprinting results in similar performance for all benchmarks, however, the penalty is slightly higher for the benchmarks that consume lower power. *Blackscholes* and *x264* are examples with normalized running times of 3.54 and 3.91, respectively. The duty cycle is determined based on the highest power application (*blackscholes* in our case). Thus, the lower power application *x264* spends more time in the rest mode than needed, leading to higher performance degradation.

In comparison to the policies discussed above, *sprint pacing* and *modified sprint pacing* provide better performance. However, *sprinting pacing* results in temperature violation up to 60% of the time (See Figure 5). On the other hand, *modified sprint pacing*, which is a thermally-aware version of *sprint pacing*, does not perform as well. This is because it does not use the intermediate V/f settings, and it exploits only 50% of the PCM capacity at the highest V/f level.

Reactive DVFS performs better than all prior sprinting strategies. The main reason is that the prior strategies primarily use the number of threads as their control knob and operate with either 1 or 16 threads, while *reactive DVFS* always runs with 16 threads if not idle. *Modified sprint pacing* also applies

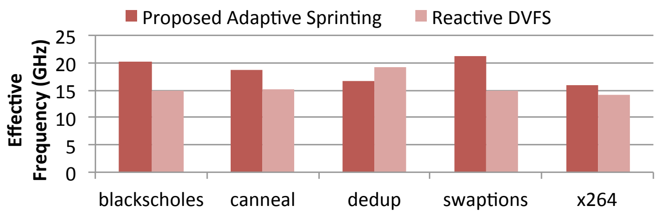


Fig. 6: Average effective frequency ($f(GHz) \times \#threads$) for the *reactive DVFS* and *adaptive sprinting* policies.

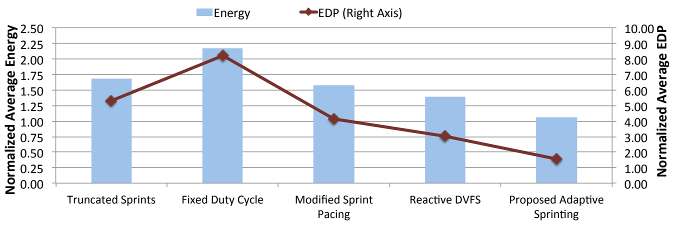


Fig. 7: Average energy and EDP normalized to the no management case.

DVFS, but it uses either the highest or the lowest V/f level, thus, cannot outperform *reactive DVFS*.

Comparison of *reactive DVFS* and *adaptive sprinting* policies show that, our proposed *adaptive sprinting* policy provides 29% shorter running time on average. Figure 6 explains the reason by comparing the average effective frequencies of the two policies. Effective frequency is the sum of the frequencies of the active cores (e.g., maximum is $2.1GHz \times 16$). Merely applying *reactive DVFS* cannot mitigate the temperature problem due to the *dark silicon* phenomena. Thus, once the thermal headroom is exhausted, *reactive DVFS* has to put the cores to idle, leading to lower effective frequency. On the other hand, *adaptive sprinting* allows extended sprints with fewer cores and avoids idling, providing up to $6.38GHz$ higher effective frequency. Our policy provides 29% and 42% higher the performance on average compared to the *reactive DVFS* and *modified sprint pacing*, respectively, without exceeding thermal limits.

Figure 7 shows the average energy and EDP values normalized to the no management case for the thermally-aware policies. *Adaptive sprinting* saves energy by 22% and 32% on average in comparison to the *reactive DVFS* and *modified sprint pacing* policies, respectively. It also provides 43% and 59% lower EDP on average compared to the *reactive DVFS* and *modified sprint pacing* policies, respectively.

VI. Conclusion

In this paper, we have proposed *adaptive sprinting* policy for efficient runtime management of systems with PCM. By monitoring the PCM state of the cores, our policy dynamically decides on the number, location and the V/f setting of the sprinting cores. It allows extended sprinting by exploiting the PCM capacity fully, thus provides substantial performance improvement. *Adaptive sprinting* policy gets closer to ideal performance than previously proposed sprinting policies while meeting the thermal constraints. Experimental evaluation shows that our policy improves performance by **29%**, saves energy by **22%**, and reduces EDP by **43%** on average compared to the best performing sprinting strategy.

References

[1] M. Shafique, S. Garg, J. Henkel, and D. Marculescu, "The EDA challenges in the dark silicon era," in *Design Automation Conference (DAC)*, pp. 1–6, June 2014.

[2] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissmann, "Power-management architecture of the intel microarchitecture code-named sandy bridge," *Micro, IEEE*, vol. 32, pp. 20–27, March 2012.

[3] A. Nussbaum, "AMD Trinity APU," 2012.

[4] A. Raghavan *et al.*, "Computational sprinting," in *HPCA*, pp. 1–12, 2012.

[5] A. Raghavan *et al.*, "Computational sprinting on a hardware/software testbed," in *ASPLOS*, pp. 155–166, 2013.

[6] A. Tilli, A. Bartolini, M. Cacciari, and L. Benini, "Don't burn your mobile!: safe computational re-sprinting via model predictive control," in *CODES+ISSS*, pp. 373–382, 2012.

[7] L. Shao *et al.*, "On-chip phase change heat sinks designed for computational sprinting," in *Semiconductor Thermal Measurement and Management Symposium*, pp. 29–34, March 2014.

[8] E. Alawadhi and C. H. Amon, "PCM thermal control unit for portable electronic devices: experimental and numerical studies," *IEEE Transactions on Components and Packaging Technologies*, vol. 26, pp. 116–125, March 2003.

[9] F. Tan and S. C. Fok, "Thermal management of mobile phone using phase change material," in *Electronics Packaging Technology Conference*, pp. 836–842, 2007.

[10] D. Yoo and Y. Joshi, "Energy efficient thermal management of electronic components using solid-liquid phase change materials," *IEEE Transactions on Device and Materials Reliability*, vol. 4, no. 4, pp. 641–649, 2004.

[11] A. Stupar, U. Drofenik, and J. Kolar, "Application of phase change materials for low duty cycle high peak load power supplies," in *International Conference on Integrated Power Electronics Systems (CIPS)*, pp. 1–11, 2010.

[12] S. Lingamneni, M. Asheghi, and K. Goodson, "A parametric study of microporous metal matrix-phase change material composite heat spreaders for transient thermal applications," in *ITHERM*, May 2014.

[13] F. Kaplan *et al.*, "Modeling and analysis of phase change materials for efficient thermal management," in *International Conference on Computer Design*, Oct 2014.

[14] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture," in *ISCA*, pp. 2–13, 2003.

[15] P. Conway *et al.*, "Blade computing with the AMD Opteron Processor $(magny-cours)$." http://bit.ly/1LVbJXn, Aug. 2009.

[16] N. L. Binkert *et al.*, "The M5 simulator: Modeling networked systems," *IEEE Micro*, vol. 26, pp. 52–60, July 2006.

[17] S. Li *et al.*, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *MICRO*, pp. 469–480, 2009.

[18] S. Thoziyoor *et al.*, "CACTI 5.1," tech. rep., April 2008.

[19] S. Park *et al.*, "Accurate modeling of the delay and energy overhead of dynamic voltage and frequency scaling in modern microprocessors," *TCAD*, vol. 32, pp. 695–708, May 2013.

[20] A. K. Coskun, R. Strong, D. M. Tullsen, and T. S. Rosing, "Evaluating the impact of job scheduling and power management on processor lifetime for chip multiprocessors," in *SIGMETRICS*, pp. 169–180, 2009.

[21] C. Bienia, *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[22] C. Hankendi, S. Reda, and A. Coskun, "vcap: Adaptive power capping for virtualized servers," in *International Symposium on Low Power Electronics and Design*, pp. 415–420, Sept 2013.

[23] S. Reda, R. Cochran, and A. Coskun, "Adaptive power capping for servers with multithreaded workloads," *Micro, IEEE*, vol. 32, pp. 64–75, Sept 2012.

[24] R. Cochran and S. Reda, "Spectral techniques for high-resolution thermal characterization with limited sensor data," in *DAC*, pp. 478–483, 2009.