Work-In-Progress: Cooling by Core-idling: Thermal-aware Thread Scheduling for Mobile Multicore Processors

Srijeeta Maity[1], Anirban Ghose[1], Soumyajit Dey[1], Sangyoung Park [2], and Samarjit Chakrabarty [3]
[1]IIT Kharagpur, [2]TU Berlin, [3]UNC Chapel Hill

Abstract—**Thermal efficient resource mapping and scheduling techniques are particularly important for mobile processors because of limited opportunities for external cooling. In mobile processors such as the ones using ARM's big.LITTLE architectures, the cores of either the big or the LITTLE processor cannot be individually voltage/frequency scaled. However, we show that by forcing all the application threads to a single core, and not having any workload on the other cores of a processor, there is still considerable thermal benefit. This is counter intuitive since all the cores run at the same frequency. We show real measurements and discuss what impact this has on thermal-aware scheduling for such multicore processors.**

I. Introduction

As processor geometries continue to shrink, thermal issues degrade the performance of embedded systems thus posing a major reliability challenge. This is evident in smartphones, where prolonged usage of compute-intensive tasks such as games result in overheating, and occasional shutdowns. Hence, thermal efficient resource mapping and scheduling techniques are of prime importance in heterogeneous embedded mobile platforms, where the scope of external cooling is limited by space constraints. So, several thermal management techniques, e.g., [1] have emerged for mitigating thermal constraint violations in multicore platforms.

Popular heterogeneous multicore mobile processors like the Exynos series employ the ARM big.LITTLE architecture comprising two types of CPUs (a big and a LITTLE) where each CPU is a cluster of multiple homogeneous cores. In such platforms, DVFS (dynamic voltage/frequency scaling) is supported at the cluster level i.e., the big and the LITTLE clusters can be independently voltage-frequency scaled. But all cores in the same cluster need to have the same voltage and frequency setting. Additionally, in such platforms, idle cores cannot be independently powered down. So even if any one core is executing a thread keeping the other cores idle, the idle cores in the same cluster still consume power at the same voltage and frequency setting as the non-idle core. This is why, running a CPU at a high frequency with multiple cores being idle, intuitively does not appear to be a suitable thread mapping option.

In a typical multicore thread mapping, the executions of multiple threads mapped to the same core are interleaved using some thread scheduling policy. But threads mapped to different cores execute in parallel. Among possible thread-core mappings, let us consider two extreme ones. ***Scenario 1:*** the threads are mapped for the best possible load balancing, and ***Scenario 2:*** all the threads are mapped onto a single core, with the remaining cores being idle but running at the same voltage and frequency as the loaded core. The load balanced mapping engages all cores for possibly the shortest time, while the other mapping engages the single core for a longer time.

Main result: Intuitively, it seems that at high clock frequencies, engaging all cores in the cluster for a shorter time would lead to better thermal behavior, compared to forcing all threads to a single core and prolonging the total runtime. However, our main result in this paper is a different finding. Our studies show that engaging a single core for longer time by forcing all the threads to it, while keeping the other cores in the cluster idle, can lead to significant peak temperature reductions. While non-intuitive, this is because of the idle cores experiencing less switching activity and therefore consuming less power, resulting in lower temperatures. With this key observation, the main contributions of this work are (i) an experimental study of the thermal benefits of different thread-to-core mappings, and (ii) a proposal on how our observations open up a new control surface for thermal-aware scheduling for mobile processors.

After discussing related work in the next section, we discuss our experimental setup, followed by outlining in Section IV how this new control surface for thermal management of multicore processors exposes new trade-offs beyond what is obtained using DVFS alone.

II. Related work

There exists a myriad of thermal-aware scheduling techniques for single-core and multicore CPU systems that primarily use frequency scaling to reduce power consumption [2] and temperature [1]. Recent research on scheduling time sensitive tasks focused on DVFS for thermal management while meeting timing constraints for uni-processor [3] and multiprocessor platforms [4]. Apart from DVFS, mechanisms like thermal-aware core mapping and inserting idle periods have also gained traction for dynamically regulating runtime chip temperatures. Prior works include (i) heuristics [5], [6], (ii) control theoretic mapping schemes [7], [8], and (iii) using Machine Learning (ML) [9], all of which use DVFS techniques [1], [10], thread migration schemes [11], thread-to-core mapping [10], [12], and their various combinations.

This work was supported by the MHRD SPARC grant "Transportation Cyber-Physical Systems".

Some dynamic thermal management techniques in multicore systems do not rely on DVFS at all and do suitable thread migration or thread-to-core assignment to decrease chip temperature, such as M-DTM [13] and RT-TAS [14]. The work reported in [13] migrates applications from big CPU cores to LITTLE CPU cores in case of thermal violation without modifying frequency settings, whereas in [14] thermally-balanced task-to-core assignments avoid simultaneous peak power dissipation on both CPUs and GPUs, thus mitigating temperature rise. But thermal management by *core idling* – as studied in this paper – has not been explored until now.

III. Setup and Motivating Example

We conducted our experiments on an ODROID XU4 platform that has a DVFS-equipped Samsung Exynos 5422 SoC comprising of (i) a quad-core ARM Cortex-A7 (LITTLE) CPU, (ii) a quad-core ARM Cortex-A15 (big) CPU, and (iii) a ARM Mali-T628 GPU. The platform supports temperature

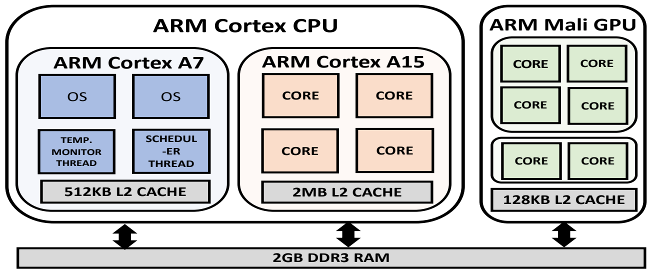

Fig. 1: Odroid XU4 architecture.

sensors for each of the A15 cores and for the Mali GPU. Like most mobile processors, the big (A15) and the LITTLE (A7) CPUs do not allow DVFS of the individual cores. For running the experiments, we have restricted the OS (Ubuntu 18.04 LTS) processes to the first two cores of the quad-core little CPU and our scheduler thread executing on the fourth core. Additionally, we have the temperature monitor process executing on the third core of the little CPU which discretely samples the temperature readings from the sensors with a sampling period of 50 ms.

In this paper, we investigate the effectiveness of different thread mapping techniques and their implications on thermal management by measuring the sensor temperature directly and characterizing the latency overhead. The plots depicted in Fig. 2 represent the temperature profiles (in $^{\circ}C$) for the individual cores in the A15 CPU while executing one iteration of a representative computation, viz., a neural network based object-detection pipeline. In this example, we consider a single layer operation where four independent instances of a General Matrix Multiply (GEMM) computation operate on matrices of size 512×512, executing in two different scenarios. Plot A represents the scenario where each instance of the GEMM computation executes in parallel on all A15 cores. Plot B represents the scenario where computation is sequentially mapped onto a single A15 core (core 1) keeping the other cores idle. All the cores were set at the maximum clock frequency setting in both the cases. We have used only the big (A15) CPU for our experiments, since it provides temperature sensors on all of its four cores.

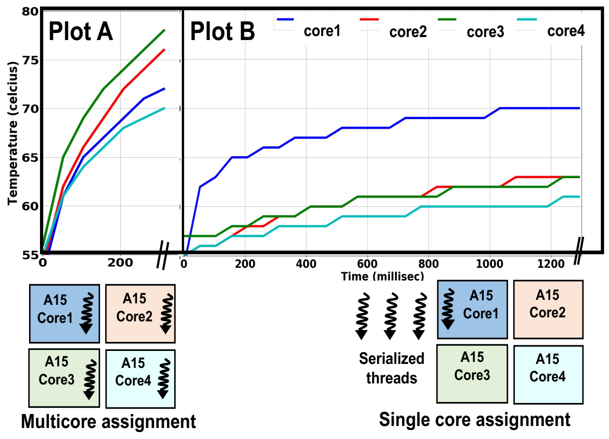

Fig. 2: Thermal profile for different assignment scenarios.

A CPU's core temperature depends on multiple factors such as (i) the difference between the steady-state temperature and the current core temperature, (ii) physical layout of the chip, (iii) the workload of the core and of the neighbouring cores, (iv) thread execution time, and the (v) the voltage frequency level [3]. We know that most of the heat dissipation on a die takes place in the vertical direction [15] and is much larger compared to lateral heat dissipation. Naturally, the scope of benefiting from lateral dissipation by having cooler adjacent cores is assumed to be negligible. Moreover, as all cores are running at maximum frequency setting throughout the execution of the threads, if a single core is engaged for a longer time it is natural to assume that the peak temperature of that core would be significantly high.

As depicted in Fig. 2, we observe that the execution time of the four instances of the GEMM operation under the multicore parallel assignment (Plot A) is approximately one-fourth of that in the single core serial assignment case (Plot B). However, the maximum core temperature is much higher in the multicore assignment (Plot A). It is also observed in Plot B that the temperature of the idle cores rises by 7 – 8 °C even though the cores are kept idle. But the temperature rise in all the four cores in Plot A is higher than the temperature rise in the cores in Plot B. Even core 1 that runs for a much longer time in Plot B has a lower peak temperature than all the cores in Plot A.

From this experiment we observe that *core idling* at cluster level can significantly drop peak temperature of CPUs even when individual core-level DVFS and power gating are not supported. We conclude that when multiple cores are engaged simultaneously, the cumulative core-to-core thermal coupling [14], [16] results in positive feedback, causing an overall temperature spike in the processor. In contrast, when only a single core is engaged, the temperature of that core initially spikes up but gradually settles down due to heat-sinking by adjacent idle cores with much lower temperature, as all cores lie on the same die through which heat flow occurs.

With an accurate characterization of the thermal impacts of such thread-to-core allocation, and a characterization of the overhead of such thread mapping, we can investigate potential thermal management techniques to leverage this new control surface. We plan to use this thread-to-core allocation with existing control-theoretic scheduling schemes [17], which have primarily focused on either tuning core-level DVFS or performing thread migration across different processors until now. We next outline a thermal-aware scheduling scheme that utilizes this control knob and demonstrates its usefulness.

IV. Proposed scheme and initial results

We propose a control-theoretic scheme for thermal-aware scheduling in embedded heterogeneous mobile platforms with thread-to-core allocation as a tuning parameter, which was not investigated before for mitigating thermal violations. Fig. 3 is an overview of a hierarchical scheduling scheme that demonstrates how the earlier discussed observations can be integrated into control-theoretic multicore task dispatch algorithms.

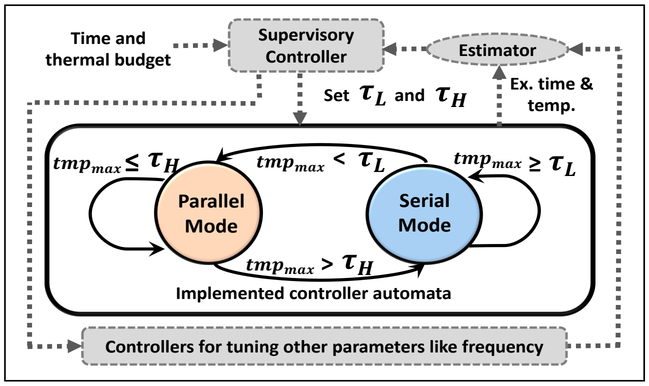

Fig. 3: Scheduler architecture: high-level overview.

As depicted in Fig.3, we implement a control scheme that switches between two distinct modes of dispatch: (i) *parallel mode*, and (ii) *serial mode*. The *serial mode* is activated if the maximum core temperature of the platform exceeds a preset threshold τ_H. In this mode, for each invocation of an application, the A15 core with the minimum temperature is identified and all threads of the application are executed sequentially on that core. This results in a drop in the maximum core temperature, as illustrated in the motivational example. Whenever the maximum core temperature falls below a threshold τ_L, the controller switches to the *parallel mode*, where threads are mapped to *all* the A15 cores. In this mode, the maximum core temperature begins to rise up again till reaching the threshold τ_H and switches back to serial mode again. The threshold parameters in this context typically control the dominance of the dispatch modes. If the parallel mode is dominant, then execution time reduces and temperature constraints are neglected, whereas the serial mode will perform better thermally at the cost of execution time.

We consider 20 consecutive iterations of the earlier workload and present a comparative evaluation between three different scheduling schemes: (i) *concurrent dispatch* which always uses parallel mode, (ii) *sequential dispatch* which always uses serial mode, and (iii) *hybrid dispatch* which is our thermal-aware adaptive scheme that switches between the parallel and the serial modes in order to achieve a suitable trade-off between thermal behavior and response times.

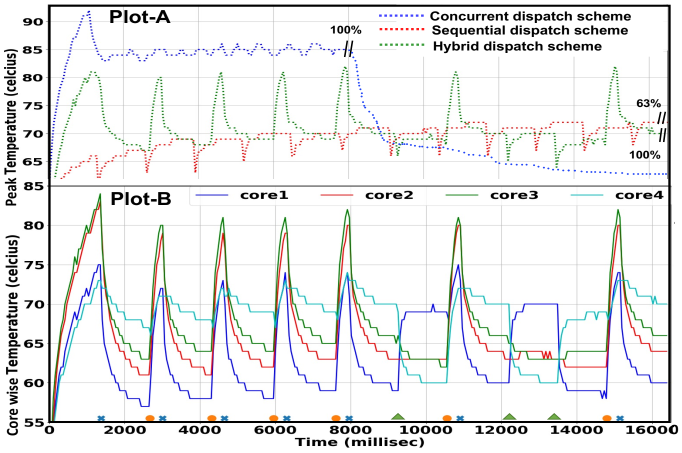

Fig. 4: Thermal profile for different dispatch schemes.

For hybrid dispatch, the threshold parameters we have considered are $\tau_H = 80°C$ and $\tau_L = 60°C$ with core frequency for all A15 cores set to 2GHz. These threshold parameters were set empirically through experiments, and they determine the condition for switching between the two modes. The peak core temperatures observed for each scheme is depicted in Plot A of Fig. 4. When using concurrent dispatch, the temperature of one of the cores increases beyond $90°C$, with the CPU governor automatically reducing the big CPU frequency to 1.6 GHz by DVFS (potentially compromising the performance in terms of latency). As a result of this the temperature is reduced to $85°C$. This change in frequency by DVFS is a safety mechanism by the hardware to avoid system failure and is known as thermal throttling. For sequential dispatch, the maximum core temperature drops after regular intervals of time as a result of control actions in the serial mode that periodically remap the threads to the core with the lowest temperature (thus reducing the average operating temperature as well). This is depicted in Plot A of Fig. 4. We observe that the execution time of sequential dispatch is increased by approximately 3.25× of that of concurrent dispatch. This difference in execution time may be attributed to the reduction in parallelism in sequential dispatch and frequency reduction due to thermal throttling in concurrent dispatch.

Our scheduling scheme, *hybrid dispatch*, provides a relatively better trade-off with respect to the other schemes. The maximum temperature always remains below $82°C$ (a 12% reduction) and the average temperature is 5% lower while taking approximately 2× more time than that of concurrent dispatch. We can further improve the latency performance by slightly alternating the current control action. In the serial mode, tasks or threads can be mapped to two diagonal cores instead of one thus balancing between fully serial and fully parallel execution. Using temperature feedback and this setting inside our hybrid dispatch scheme, the peak and average temperatures reduce by 8% and 3% respectively, while taking approximately

1.7× more time than that of concurrent dispatch. Hence, in general we can dynamically map threads to any $1 \leq i \leq n$ number of cores (for an n-core system) with an appropriate control strategy.

Plot B in Fig. 4 depicts a detailed thermal profile of the individual A15 cores with hybrid dispatch. Instances of alternating between the parallel mode and the serial mode and vice-versa are marked by blue crosses and orange dots, respectively. Instances depicting the remapping of the active core in serial mode are marked by green triangles in the same plot. At these instances, the temperature of the hottest core starts falling rapidly as it becomes idle and the temperature of the newly assigned core rapidly increases as the threads start running on it. This occurs only when the scheduling scheme operates in serial mode for a prolonged period of time. Alternating between different modes of dispatch during the application's lifetime only causes temperatures of only two A15 cores (core 2 and core 3) to rise significantly. The maximum temperatures of these cores exceed $80°C$ for a very short time and never crossed $90°C$, thus inhibiting the activation of DVFS by the CPU governor. The remaining core temperatures always remain below $75°C$.

For mobile applications [17] that run for prolonged time periods in the background such as GPS updates, system and application specific updates, and device synchronisation, power drainage and thermal violation are of greater concern than their response or execution times. Maintaining stable chip temperatures and power dissipation rates are critical for executing such applications in mobile devices. This is where our proposed scheduling scheme is suitable. In this work, we have highlighted only one hybrid dispatch mode for scheduling. We note that this can be further refined by incorporating other control surfaces such as frequency scaling, and also target metrics that enforce deadline requirements and performance guarantees.

To account for workload-awareness and the dynamic behaviour of workloads, we propose to add a top-level controller as an extension to this work. This top level controller can set the *goal priority* of the system based on its current performance. If the system misses timing constraints, the priority will be changed to reduce execution time at the cost of higher core temperatures. But if the maximum core temperature goes beyond the thermal envelope, reducing the core temperature is given priority at the cost of increased execution time.

The proposed approach promotes *opportunistic serialization* as a control surface which can be a useful augmentation to existing control theoretic as well as machine learning based multicore scheduling schemes [9]. Existing works that advocate using a mix of machine learning and control theory for heterogeneous system scheduling [9] have shown that the right combination of techniques from these two domains can provide good operating points along with guaranteed performance. We propose the use of a Supervisory Controller (SC) as the top level controller in the hierarchical scheduler in Fig. 3. Given a timing budget and temperature envelope, SC can set the threshold parameters empirically and refine them on the fly at different iterations of the application. The values of τ_H and τ_L control the dominance of the two modes, which in effect tunes the priority of conflicting control objectives, viz., temperature and execution time. The SC can periodically act upon system state estimates generated using temperature, execution time measurements, and core frequency settings and perform periodic high level parameter updates for the thermal controller (i.e., τ_H and τ_L), DVFS controller, DRAM frequency controller, etc. The SC and other control surfaces are shown as dotted components in Fig. 3 and are the future extension of this current work.

V. Concluding Remarks

The goal of this paper was to present a new thermal-aware adaptive scheduling scheme based on the observation that restricting application level parallelism can impact the thermal behavior of multicore processors. Our current efforts include: (i) realizing the hierarchical scheduling scheme for multicore CPU-GPU platforms, (ii) adding support for data-parallel languages like OpenCL/Vulkan API, and (iii) quantifying the impact on user experience, viz., response time guarantees of the proposed approach, versus existing approaches that rely on DVFS alone.

References

[1] H. F. Sheikh *et al.*, "An overview and classification of thermal-aware scheduling techniques for multi-core processing systems," *Sustainable Computing: Informatics and Systems*, vol. 2, no. 3, pp. 151–169, 2012.

[2] N. Peters *et al.*, "Web browser workload characterization for power management on hmp platforms," in *CODES+ ISSS*, 2016.

[3] Y. Lee *et al.*, "Thermal-aware resource management for embedded real-time systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2857–2868, 2018.

[4] K. Lampka *et al.*, "Keep it slow and in time: Online dvfs with hard real-time workloads," in *DATE*, 2016.

[5] S. Maity *et al.*, "Thermal-aware adaptive platform management for heterogeneous embedded systems," in *EMSOFT*, 2021.

[6] S. Maity *et al.*, "Thermal load-aware adaptive scheduling for heterogeneous platforms," in *VLSID*, 2020.

[7] A. Bartolini *et al.*, "Thermal and energy management of high-performance multicores: Distributed and self-calibrating model-predictive controller," *IEEE TPDS*, vol. 24, no. 1, pp. 170–183, 2013.

[8] A. M. Rahmani *et al.*, "Spectr: Formal supervisory control and coordination for many-core systems resource management," in *ASPLOS*, 2018.

[9] N. Mishra *et al.*, "Caloree: Learning control for predictable latency and low energy," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 184–198, 2018.

[10] B. K. Reddy *et al.*, "Inter-cluster thread-to-core mapping and dvfs on heterogeneous multi-cores," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 4, no. 3, pp. 369–382, 2017.

[11] V. Hanumaiah *et al.*, "Performance optimal online dvfs and task migration techniques for thermally constrained multi-core processors," *IEEE TCAD*, vol. 30, no. 11, pp. 1677–1690, 2011.

[12] N. Peters *et al.*, "Phase-aware web browser power management on hmp platforms," in *ICS*, 2018.

[13] Y. G. Kim *et al.*, "M-dtm: Migration-based dynamic thermal management for heterogeneous mobile multi-core processors," in *DATE*, 2015.

[14] Y. Lee *et al.*, "Thermal-aware scheduling for integrated cpus–gpu platforms," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, pp. 1–25, 2019.

[15] W. Huang *et al.*, "Hotspot: A compact thermal modeling methodology for early-stage vlsi design," *IEEE Transactions on very large scale integration (VLSI) systems*, vol. 14, no. 5, pp. 501–513, 2006.

[16] S. Corbetta *et al.*, "Estimation of thermal status in multi-core systems," in *ISCAS*, 2011.

[17] B. Dietrich *et al.*, "Time series characterization of gaming workload for runtime power management," *IEEE Transactions on Computers*, vol. 64, no. 1, pp. 260–273, 2013.