DeadPool: Performance **Dead**line Based Frequency **Pool**ing and Thermal Management Agent in DVFS Enabled MPSoCs

Somdip Dey[1,3], Amit Kumar Singh[1], Xiaohang Wang[2], and Klaus Dieter McDonald-Maier[1]
[1]*University of Essex, U.K.*
[2]*South China University of Technology, China.*
[3]*Samsung R&D Institute, U.K.*
Corresponding E-mail: somdip.dey@essex.ac.uk

***Abstract*—High operating temperature and frequent thermal cycles in a multi-processor system-on-chip, which is now popularly utilized in mobile/Edge devices, harm the overall lifespan and reliability of such devices. In this paper, we propose an intelligent software agent that works alongside other resource mapping and partitioning mechanism in order to monitor and reduce the operating temperature of the system by regulating the operating frequency of the CPU cores while catering for performance constraint at the same time. Our proposed approach, DeadPool thermal management agent, is able to reduce the overall operating temperature of the system by 24.21% and reduce thermal cycle by 67.42% at the most when compared to the state-of-the-art methods.**

***Index Terms*—MPSoC, DVFS, agent, performance deadline, thermal management**

I. Introduction and Motivation

In recent times we could notice an extensive use of heterogeneous Multi-Processor Systems-on-Chips (MPSoCs) [1]–[4] in embedded devices, which employ several types of processing elements within a single chip to deliver power as well as energy efficient computing capabilities. A lot of research is performed to develop algorithms that cater for performance [5]–[7], but comparatively less amount of research could be found which focuses on developing algorithms that cater for temperature-aware energy efficient allocation of computing resources in such embedded devices while meeting end-user demands for performance.

Elevated operating temperatures and thermal cycling on the device have adverse effects on Integrated Circuits (ICs) and reliability of electronic products can be heavily influenced by spatial or temporal gradients or absolute temperatures [8]–[10]. Here, thermal cycling is the phenomenon of the operating temperature increasing from an initial value and then decreasing to the starting value periodically, and each occurrence of the phenomenon is signified as one thermal cycle. To mitigate reliability issues as a result of increased operating temperature and thermal cycling most mobile devices often come with thermal capping. Nowadays devices often come with hard-wired Thermal Management Units (TMUs) as well as software TMUs which regulates the energy consumption as well as the temperature of the associated components. However, so far none of these state-of-the-art thermal management units have been proven to be effective enough to cater for performance, energy efficiency as well as thermal-regulation at the same time.

Majority of the published noteworthy studies [10]–[13] were implemented and experimented in simulations such as Sniper [14] multicore, Synopsys software, etc. instead of experimenting on real devices, which could differ from the practical results achieved from real devices. In two other studies [15], [16], the depicted methods involve predicting the future core temperatures to adjust the workloads or frequencies before exceeding a set threshold, but computation of such predictive temperatures increase the performance overhead of such methods. In a different study [9], the researchers proposed a thermal management mechanism implemented on Q-Learning based reinforcement learning (RL) strategy to optimize thermal behavior while improving performance, but such a method has its own limitations. In Q-Learning based RL [9], the mechanism takes a decision (action) and profiles the outcome (state), and then keep a record of the actions and states in a table called, "Q-table" to learn from it consecutively. However, Q-Learning based mechanisms incur an overhead of traversing the table and learning from it, especially if the table contains a lot of actions and states, while catering for thermal reduction objective. Hence, the specific challenges this paper tries to address is as follows:

1) Design a light-weight thermal management mechanism, which is intelligent enough to cater for performance as well as maintain a desirable temperature of the cores.
2) The mechanism has to be self-adapting as the device ages due to degradation in the chemical properties of the device itself such that the thermal management mechanism can continue being effective with time.

To address the aforementioned challenges and to overcome the limitations of the existing approaches we propose an agent based thermal management mechanism called DeadPool TMA: Performance **Dead**line Based Frequency **Pool**ing and **T**hermal **M**anagement **A**gent, which is capable of catering for both performance and reduced operating thermal behavior of the device. DeadPool TMA or in extended abbreviation, DeadPool is a software agent that monitors the operating temperature and then selects the appropriate frequency from the pool of available frequencies so that temperature is low while catering for performance deadline profiled from previous executions of the same application. Our DeadPool agent is able to outperform the state-of-the-art thermal management mechanism even after being simple in design. To this end, this paper makes the following contributions:

1) A thermal management mechanism to monitor and reduce temperature of the CPU cores to improve the relia-

bility of the system without compromising performance of the executing applications.

2) Implementation and validation of the proposed thermal management intelligent agent on a real hardware platform, the Odroid-XU4.

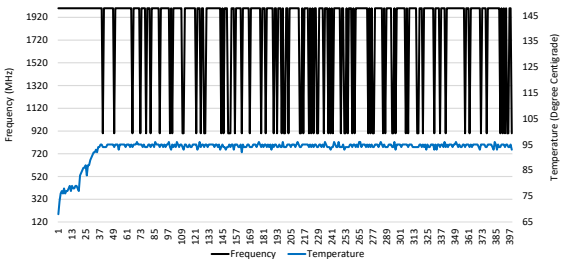


Fig. 1. Frequency and Temperature over the execution period (secs) of Streamcluster using Linux's ondemand governor

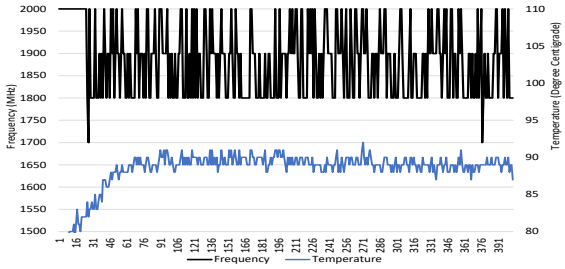


Fig. 2. Frequency and Temperature over the execution period (secs) of Streamcluster using DeadPool TMA

A. Motivational Analysis

We present a couple of empirical studies to motivate the need for a thermal management strategy, which is capable of catering for performance and thermal requirement at the same time, such as the case of DeadPool.

1) Frequency scaling and thermal behavior: An important observation, which is worth mentioning is that when we executed a mixed load (compute intensive as well as memory intensive) program on the ARM Cortex A-15 CPU cores on an Odroid XU4 [17] platform, which houses the ARM's big.LITTLE architecture (4 ARM Cortex A-15 (big) CPUs and 4 ARM Cortex A-7 (LITTLE) CPUs) and 6 ARM Mali-T628 GPU, the core temperature rises very fast due to executing workloads that are both compute and memory intensive. Fig. 1 graphically represents the operating frequency and operating temperature (in °Centigrade) of the big core, which has the worst temperature behavior[1] on Odroid XU4 while executing Streamcluster[2] from the PARSEC benchmark suit [18] on Linux OS, running with Ondemand governor power scheme. In Fig. 1, the horizontal axis represents the execution time steps whereas the vertical primary axis (left-hand side) reflects the operating frequency (in MHz) of the big cores on which the program is executing and the vertical secondary axis represents the operating temperature (in °Centigrade) of the big core with worst temperature behavior. Upon close inspection we could notice that the maximum temperature reached amongst the ARM big cores is 95°Centigrade with an execution time of 433.12 secs while executing Streamcluster with *native* option on the ondemand governor. Since the CPU temperature reaches 95°Centigrade, the default thermal management unit (TMU) of the OS under ondemand governor starts to regulate the operating frequency and as a result, reduces the temperature of the cores by CPU throttling[3]. Although CPU throttling is good in terms of reliable operations but the stock thermal management algorithm of the OS is not intelligent enough to provide performance at the same time. In Fig. 1 we could notice that the clock speed varies from 2000 MHz to 900 MHz over time, which causes a drop in performance of the executing application(s). In comparison, when we utilize DeadPool, which adheres to the strict performance and temperature deadline collected through offline/online profiling (of Streamcluster), while executing Streamcluster in *native* mode, we are able to achieve 7.44% (see Fig. 4) reduction in average operating temperature (in °Centigrade) of the big core, which has the worst temperature behavior during the execution, yet taking the same execution time as the ondemand governor. While executing Streamcluster on DeadPool the performance deadline was set to be the execution time of the program while running on the ondemand governor.

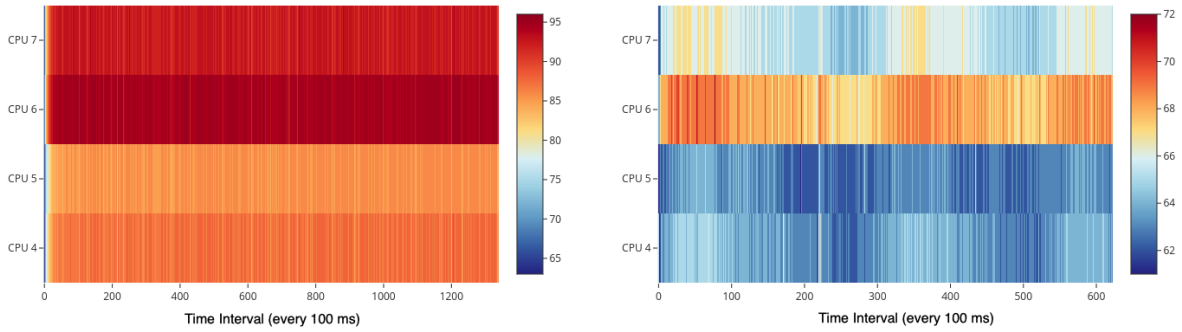
2) Reduction in spatial and temporal thermal gradient: In another experiment, we executed Blackscholes benchmark from the PARSEC [18] benchmark suit on the big CPU cores of Odroid XU4, which we denote as CPU 4, 5, 6 and 7 consecutively in our experiments, using different thermal management strategies including DeadPool. When we executed Blackscholes on the 4 big CPUs using performance governor it took 91.302 secs and peak operating temperature of all 4 big CPUs reached close to 95°C, which is reflected in Fig. 3.(a). Fig. 3.(a) shows the variation in operating temperature of the 4 big CPUs over time (reading taken every 100 ms). Whereas, when we executed Blackscholes using DeadPool while having the execution time (performance) deadline set to the execution time taken during the performance governor, we are able to reduce the overall operating temperature by 24.21% while catering for performance (see Fig. 3.(b)). Comparing the results of Fig. 3.(a) and 3.(b), we could certainly notice that DeadPool is also capable of reducing spatial and temporal thermal gradient of the CPUs, and not just the overall peak temperature. Here, the term "thermal gradient" signifies the physical quantity that describes in which direction and rate the temperature is changing.

Therefore, the results from the aforementioned experiments proved that default scheduling strategies (governors), which come bundled with the Linux, are not capable of handling both performance of the executing program and reduce operating temperature of the device at the same time. On the other hand, DeadPool is able to cater for performance requirement while reducing spatial and thermal gradient of the device while executing the application.

[1]Usually the cores exhibit worse temperature behavior, which are residing close to the memory.

[2]We choose the *native* option of the Streamcluster to mimic real-world data-mining algorithms, which are both compute intensive and memory intensive.

[3]Adjusting the clock speed of the CPU to use less power consumption and thus reduce CPU temperature for improved reliability.



(a) Using Performance governor

(b) Using DeadPool TMA

Fig. 3. Operating temperature over time while executing Blackscholes on 4 ARM Cortex A-15 (big) CPUs using DeadPool TMA and Performance governor

II. Related Work

One noteworthy study in thermal management of the device, pursued by Kamal et al. [11] proposed a heuristic based thermal stress-aware mechanism for management of power and temperature in MPSoCs formulated in a convex optimization problem to reduce thermal gradient of the device. This approach was implemented on Sniper multicore simulator [14] and was able to reduce spatial and temporal thermal gradients by 7% and 18% respectively. In a different study, Iranfar et al. [9] proposed a thermal management mechanism utilizing heuristic based reinforcement learning (RL). In [9], the mechanism uses thread migration and DVFS as actions of Q-Learning based RL to optimize power and thermal gradient while improving performance, and Sniper multicore simulator is utilized to prove the efficacy of the mechanism, which was able to reduce thermal cycle by 39%, temporal thermal gradient by 34% and spatial thermal gradient by 21%. In another work by Iranfar et al. [10], the researchers proposed a multi-tier hierarchical thermal stress-aware power and temperature management framework for MPSoCs, where the methodology used similar convex optimization solution in multi-layer to improve mean time to failure (MTTF)[4]. The effectiveness of this methodology is again proved based on simulations performed on Sniper multicore simulator. All the aforementioned noteworthy studies were implemented and experimented in simulations[5] instead of experimenting on real devices, which could differ from the practical results achieved from real devices because there are so many other contributing factors that account for the operating temperatures such as ambient surrounding temperature, chemical reactions on the devices due to weather change, etc.

Moreover, [10], [11], [14] do not actively cater for performance while actively catering for thermal gradient reduction on the device whereas, [9] does focus on improving performance of the executing application while optimizing operating thermal behavior of the device but Q-Learning based reinforcement learning approaches have their own limitation. In Q-Learning, convergence of the reinforcement learning is dependent on the number of actions and states in the Q-table. Since, thermal behavior on mobile devices for applications is not constant and dependent on many factors such as computation being performed at the moment, size of the data for computation, ambient temperature, other interrupt services which are capable of changing the work-flow of executing application, and hence, the size of Q-table could be large to solve convergence of learning after incorporating all the possible actions and states. Therefore, the size of Q-table could contribute to an overhead of taking an action, but thermal strategies on the device require a more light weight learning mechanism to overcome such limitations.

In our approach, we utilize the concept of intelligent software agents and linear regression based semi-supervised machine learning to design a light-weight fast heuristic based operating temperature regulator, which could be used on top of any scheduler or governor or any other types of system resource manager to achieve efficient operating temperature without sacrificing performance requirement of the executing applications.

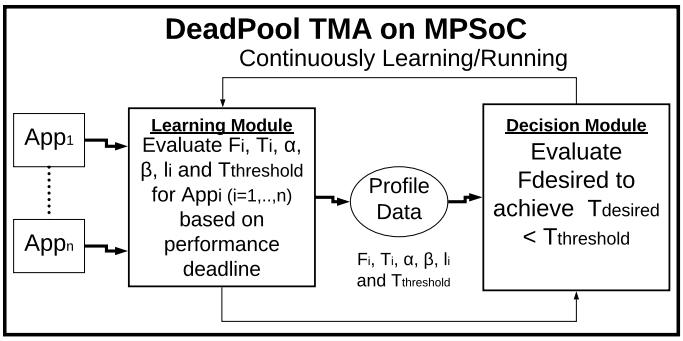


Fig. 4. Block diagram of DeadPool thermal management agent

III. Proposed Methodology: DeadPool

There are four schools of thought for artificial intelligence [19]: Thinking Rationally [20], Acting Rationally [21], Think-

[4]Mean time to failure (MTTF) is the length of time a device is expected to operate/last till failure.

[5]Although modern simulators such as Sniper multicore provides simulation results very close to real devices but there are many unaccountable factors that could happen when executed on a real device instead of simulation, especially in case of temperature variance.

ing Humanly [22] & Acting Humanly [23], where intelligent agents are built or designed to portray each school of thought. In our approach, DeadPool, we propose an intelligent software agent that is a continuous running application and sits in the application layer of the OS to monitor the operating temperature of the system while executing an application. DeadPool controls the thermal behavior of the system by *Thinking and Acting Rationally* based on our heuristic approach and works along side existing resource manager, which effectively maps and allocates tasks to the CPUs. Over time it creates a relationship between the operating temperature of an application and the operating frequency of the CPUs taking leverage of DVFS capabilities of the MPSoC. The DeadPool has two modules in it: *Learning* and *Decision* Modules. Fig. 4 shows the block diagram of the agent.

Assumptions: Since, majority of the modern mobile/Edge devices utilize cluster wise DVFS, DeadPool's implementation is written taking cluster wise DVFS into perspective. If we assume $P_{threshold}$ is the performance deadline by which the executing application has to be finished and T_{max} is the maximum operating temperature of the device while catering for P, and $T_{threshold}$ is the maximum operating temperature, which is set as the thermal cap that the system should not exceed, of the cores in cluster, then our methodology applies for cases where $T_{max} \leq T_{threshold}$.

A. Learning Module

In the *Learning Module* the DeadPool first gets the performance deadline[6] ($P_{threshold}$) and the operating temperature threshold ($T_{threshold}$) for an executing application. Here, the value of $T_{threshold}$ is less than the default thermal cap set in Linux and the main objective of DeadPool is to execute the program at a frequency which will adhere to $P_{threshold}$ and $T_{threshold}$. $P_{threshold}$ for an executing application is either user-provided or is fetched by profiling the application in advance by executing it several times and the best performance out of all the profiling execution is noted and fed to DeadPool as $P_{threshold}$. Then DeadPool reduces the operating frequency of the CPUs to next lower frequency level step and records the operating temperature as well as the operating frequency such that a regression equation (see Eq. 1) could be formed. After using the Eq. 1, it evaluates the value of α over time. In Eq. 1 F_i is the operating frequency at time instance i, T_i is the operating temperature at the same time instance, α is the relationship variable and β is the error defining variable. Now, for each instance of operating frequency (F_i) the agent maps it to an instance of a peak operating temperature of the cluster cores (T_i) ($F_i \longmapsto T_i$) as well as performance (P_i) of the executing application. Therefore, operating frequency instance could be mapped to a tuple of peak operating temperature and performance instance ($F_i \longmapsto < T_i, P_i >$). Here, the peak operating temperature is the maximum operating temperature of the core among all the other cores in the cluster. By reducing the frequency (using DVFS capabilities) by a certain number of frequency scaling levels[7], different F_i, T_i, α, β & frequency scaling level reduction steps (l_i) are recorded by the agent, which would be used to decide which frequency to drop to in the Decision module. In contrary to majority of Q-Learning based thermal management schemes where all the actions and states have to be recorded for convergence of learning, in DeadPool only the values of α and β have to be stored, since DeadPool can compute the predicted operating temperature in the future by using those values (see Decision module - Sec. III-B).

DeadPool continues to reduce the frequency ($F_i - l_i =< T_i, P_i >$)(using Eq. 2) and maps frequency to temperature until the performance deadline for the executing application is met i.e. $P_i \leq P_{threshold}$. If performance deadline is not met i.e. $P_i \geq P_{threshold}$ then it stops reducing the frequency and increases it to the next frequency scaling level in contrary ($F_i + l_i =< T_i, P_i >$). However, when the temperature starts to rise up again i.e. $T_i \rightarrow T_{threshold}$, the agent reduces the frequency as mentioned above while meeting the performance deadline i.e. $P_i \leq P_{threshold}$. DeadPool tracks the instantaneous operating temperature and performance of the application periodically and this is called as the control period. For our implementation the control period is set to 100 ms and hence in every 1 second DeadPool checks the operating temperature and performance 5 times so that it could learn and decide the next action to take. However, 100 ms control period suits the set of applications being used for our device and could be set to any value as the user sees fit for their system but it has to be kept in mind that a higher value of control period means the agent is less responsive in learning and taking decision.

$$T_i = \alpha \times F_i + \beta \tag{1}$$

$$F_i \pm l_i =< T_i, P_i > \tag{2}$$

B. Decision Module

In the *Decision Module*, the DeadPool uses the relationship variable (α), error variable (β), frequency scaling level steps l_i computed from *Learning Module* and calculates the desired frequency ($F_{desired}$) using the following equation:

$$F_i - F_{desired} = (T_i - T_{desired})/(\alpha) \quad , \quad where \;\; T_{desried} < T_{threshold} \tag{3}$$

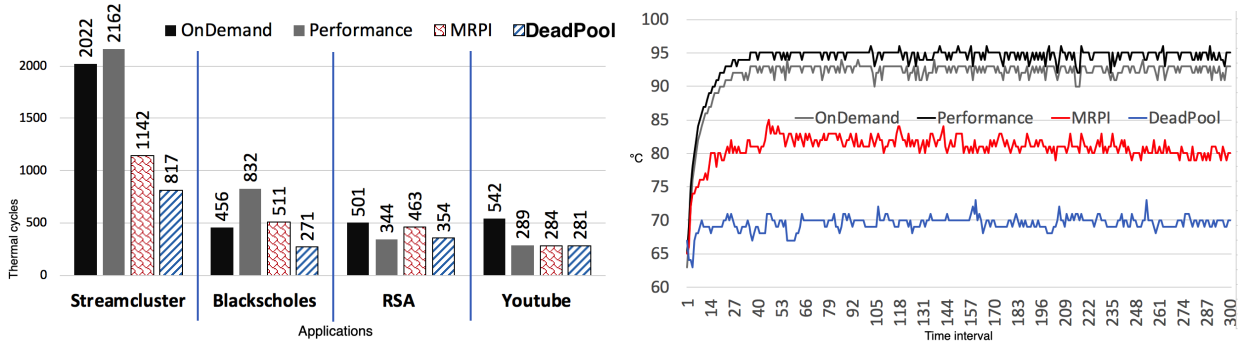
DeadPool then tries to find the least value of ($F_i - F_{desired}$) i.e. the least number of frequency scaling level, it should drop to so that the operating temperature could be reduced without affecting the overall performance of the executing application as opposed to what generic TMUs or DPTMs[8] do. Here F_i is the current operating frequency and $F_{desired}$ is the operating frequency to which the CPU should drop to in order to provide performance as well as reduced operating temperature ($T_{desired}$), which is lower than $T_{threshold}$, as opposed to current operating temperature (T_i). Therefore, from the Eq. 3 the agent has to find the least value of ($F_i - F_{desired}$) out of all n possibilities by computing the predictive operating temperature using the values of $T_{threshold}$, α_i & β_i for every frequency scaling level steps (l_i).

Example: The F_i is 2000 MHz and T_i is 75°C, but the goal is to reduce the operating temperature by 1°C, therefore, $T_{desired}$ becomes 74°C. Now after utilizing the saved value of

[6]Performance deadline depends on the type of applications. For some application it could be execution time deadline or for computer vision applications it could be desired frames per second, etc.

[7]For example ARM Cortex A-15 CPUs in the Odroid XU4 has 19 frequency scaling levels.

[8]DPTM is Dynamic Power and Thermal Management techniques that regulate both power and temperature.



(a) Thermal cycles for different applications when employing different power & thermal management schemes

(b) Maximum operating temperature (°C) over time (secs) for Blackscholes while employing different power & thermal management schemes

Fig. 5. Thermal cycle and maximum operating temperature reduction employing different methodologies for different applications

α, $F_{desired}$ is evaluated to be 1800 MHz (using Eq. 3) then the operating frequency needs to be reduced by 2 frequency scaling steps ($= \frac{2000-1800}{100}$), given the fact that the device allows each frequency scaling level steps of 100 MHz.

IV. Experimentation and Validation Results

A. Experimental Setup

We implemented our proposed DeadPool software agent in an Odroid XU4 [17] on top of the Memory Reads Per Instruction (MRPI) approach[9] based Mapping and Resource Manager [24] as well as on stock Linux Governors to perform the experiments and show the effectiveness of our methodology. Since Odroid XU4 only has temperature sensors on ARM Cortex A-15 big CPUs and Mali-T628 GPU, to validate our approach we have executed the applications on the big CPU cores referred to as CPU 4, CPU 5, CPU 6 & CPU 7, and observed the thermal variance over time. We have run all our experiments on UbuntuMate version 14.04 (Linux Odroid Kernel: 3.10.105). We executed several benchmark applications from the PARSEC [18] suit such as Streamcluster, Blackscholes, etc. We have also played Youtube videos on Chromium browser and ran RSA encryption and decryption on 512, 1024, 2048 and 4096 bits.

B. Experimental Results

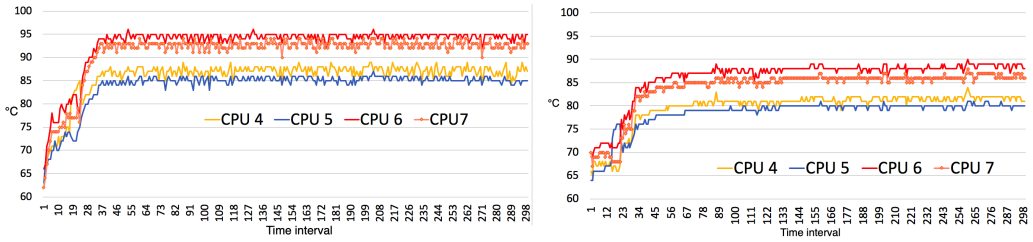
Fig. 5 shows that DeadPool was able to reduce the thermal cycle in the system by 59.59% in comparison to Linux's Ondemand governor and by 62.21% in comparison to Performance governor for Streamcluster benchmark, whereas it achieved reduction of thermal cycle by 46.496% at most in comparison to MRPI for the same. Deadpool reduced thermal cycle by 67.42% for Blackscholes benchmark in comparison with performance governor. DeadPool was also able to reduce the overall operating temperature by 24.21% (see Fig. 5.b) for Blackscholes compared to Linux's Performance governor while executing the application.

Fig. 6 shows the operating temperature values on 4 ARM Cortex A-15 big CPU cores while executing Streamcluster benchmark with *native* option and employing Ondemand governor and DeadPool. In all the figures, we have only shown fraction of the total execution period of each application because there was similar behavior in temperature during execution time. While using Ondemand governor the average temperature on on each ARM big core (CPU 4, CPU 5, CPU 6 & CPU 7) are 74°C, 73°C, 80°C, 78°C respectively and the maximum temperature on CPU 4, CPU 5, CPU 6 & CPU 7 are 86°C, 84°C, 94°C, 92°C respectively. Whereas, using the DeadPool the average temperature on each ARM big core is 81°C, 79°C, 87°C, 85°C respectively and the maximum temperature on each big core is 90°C, 81°C, 90°C, 91°C respectively. Therefore, DeadPool is able to reduce the maximum operating temperature of the big CPUs by 6.25% while executing Streamcluster over time.

Fig. 7 shows the operating temperature values on 4 ARM Cortex A-15 big CPU cores while playing Youtube video on Chromium browser by employing Ondemand governor and DeadPool. While using Ondemand governor the average temperature on each ARM big core is 74°C, 73°C, 80°C, 78°C respectively (not including the decimal digits after evaluating average) and the maximum temperature on each big core are 90°C, 86°C, 95°C, 94°C. Whereas, using the DeadPool the average temperature on each ARM big core is 75°C, 74°C, 82°C, 80°C respectively (not including the decimal digits after evaluating average) and the maximum temperature on each big core are 84°C, 80°C, 90°C, 88°C. Therefore, DeadPool is able to reduce the maximum operating temperature of the big CPUs by 5.26% while playing Youtube videos on Chromium browser over time.

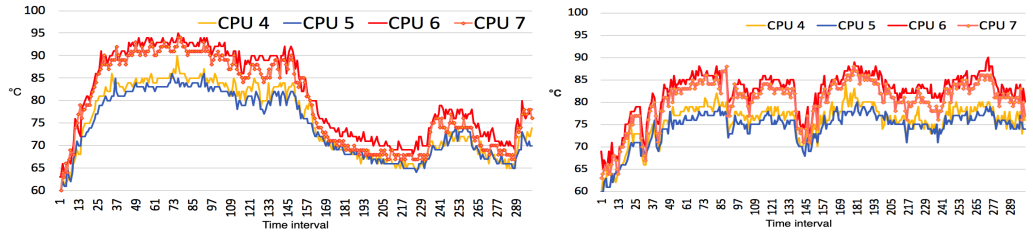
We have also compared our agent based approach with TheSPoT thermal management mechanism [10], which is able to reduce peak temperature by 24% for their heuristic approach and by 18% for their optimal approach while executing Blackscholes. TheSPoT also has a performance overhead of 4.8% and 3.5% for their heuristic and optimal approaches respectively. In comparison, DeadPool is able to achieve 24.21% reduction in temperature for Blackscholes without compromising performance and hence, outperforms the TheSPoT. On the other hand, when we compared DeadPool with the thermal management mechanism implementing heuristics based reinforcement learning (RL) [9], DeadPool was able to reduce operating temperature by 24.21% at the most and reduce thermal cycle by 62.21% at the most whereas, RL based strategy was only able to reduce the operating temperature by 21% at the most and reduce thermal cycle by 39% at the most.

[9]In the study [24], the researchers have proposed a novel workload management system, which classifies workloads of the executing applications based on MRPI metric and manages DVFS levels of cores.



(a) Ondemand governor (b) DeadPool TMA

Fig. 6. Temperature values (°C) over time (secs) on 4 ARM Cortex A-15 big CPUs while executing Streamcluster benchmark with *native* option using Ondemand & DeadPool TMA



(a) Ondemand governor (b) DeadPool TMA

Fig. 7. Temperature values (°C) over time (secs) on 4 ARM Cortex A-15 big CPUs while playing Youtube video on Chromium browser with *native* option using Ondemand & DeadPool TMA

V. Conclusion

In this paper, we proposed a light-weight intelligent software agent based mechanism that can monitor and reduce the operating temperature of the system by regulating the operating frequency of the CPU cores while catering for performance constraint of an executing application. Our proposed approach is able to reduce the operating temperature by 24.21% at most when compared to state-of-the-art resource management mechanism and also reduce the thermal cycle of the system up to 67.42% for certain applications.

Acknowledgment

This work has been supported by the UK Engineering and Physical Sciences Research Council EPSRC [EP/R02572X/1 and EP/P017487/1], the Natural Science Foundation of Guangdong Province No. 2018A030313166, and Pearl River S&T Nova Program of Guangzhou No. 201806010038. Somdip would also like to thank his colleagues from the University of Essex, the Samsung R&D Institute UK, and his parents for their support.

References

[1] S. Isuwa *et al.*, "Teem: Online thermal- and energy-efficiency management on cpu-gpu mpsocs," in *2019 Design, Automation, and Test in Europe (DATE 2019)*. IEEE, 2019.

[2] S. Dey *et al.*, "Edgecoolingmode: An agent based thermal management mechanism for dvfs enabled heterogeneous mpsocs," in *2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems*. IEEE, 2019.

[3] ——, "Mat-cnn-sopc: Motionless analysis of traffic using convolutional neural networks on system-on-a-programmable-chip," in *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. IEEE, 2018.

[4] ——, "Energy efficiency and reliability of computer vision applications on heterogeneous multi-processor systems-on-chips (mpsocs)," in *Adaptive Many-Core Architectures and Systems workshop, York, UK*.

[5] A. K. Singh *et al.*, "A survey and comparative study of hard and soft real-time dynamic resource allocation strategies for multi-/many-core systems," *ACM CSUR*, vol. 50, no. 2, 2017.

[6] ——, "Learning-based run-time power and energy management of multi/many-core systems: current and future trends," *JOLPE*, vol. 13, no. 3, 2017.

[7] B. K. Reddy *et al.*, "Online concurrent workload classification for multi-core energy management," in *DATE*, 2018.

[8] K. DeVogeleer *et al.*, "Modeling the temperature bias of power consumption for nanometer-scale cpus in application processors," in *2014 SAMOS XIV*, 2014.

[9] A. Iranfar, S. N. Shahsavani, M. Kamal, and A. Afzali-Kusha, "A heuristic machine learning-based algorithm for power and thermal management of heterogeneous mpsocs," in *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2015.

[10] A. Iranfar *et al.*, "Thespot: Thermal stress-aware power and temperature management for multiprocessor systems-on-chip," *IEEE TCAD*, vol. 37, no. 8, 2018.

[11] M. Kamal *et al.*, "A thermal stress-aware algorithm for power and temperature management of mpsocs," in *2015 DATE & Exhibition*, 2015.

[12] M. Ghasemazar *et al.*, "Robust optimization of a chip multiprocessor's performance under power and thermal constraints," in *IEEE ICCD*, 2012.

[13] H. Amrouch *et al.*, "Voltage adaptation under temperature variation," in *IEEE SMACD*, 2018.

[14] T. E. Carlson *et al.*, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulations," in *SC*, Nov. 2011, pp. 52:1–52:12.

[15] G. Singla *et al.*, "Predictive dynamic thermal and power management for heterogeneous mobile platforms," in *2015 DATE & Exhibition*, 2015.

[16] G. Bhat *et al.*, "Algorithmic optimization of thermal and power management for heterogeneous mobile platforms," *IEEE TVLSI*, vol. 26, no. 3, 2018.

[17] "Odroid-xu4," https://goo.gl/KmHZRG, accessed: 2018-07-23.

[18] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Princeton University, January 2011.

[19] M. Negnevitsky, *Artificial intelligence: a guide to intelligent systems*, 2005.

[20] D. McDermott and E. Charniak, "Introduction to artificial intelligence," *Reading: Addison-Wesley*, 1985.

[21] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*, 2016.

[22] J. Haugeland, "Artificial intelligence: The very idea. 1985," *Cited on*, 1985.

[23] E. Rich and K. Knight, "Artificial intelligence," *McGraw-Hill, New*, 1991.

[24] B. K. Reddy *et al.*, "Inter-cluster thread-to-core mapping and dvfs on heterogeneous multi-cores," *IEEE TMSCS*, 2017.