Balancing Performance and Aging in Cloud Environments

Thiago Gonçalves, Antonio Carlos S. Beck and Arthur F. Lorenzon
Institute of Informatics, Federal University of Rio Grande do Sul, Porto Alegre, Brazil
{*thiago.goncalves, caco, aflorenzon*}*@inf.ufrgs.br*

Keywords: Process Variability, Aging, Cloud Computing.

Abstract: As the number of cores per chip increases, cloud servers become more capable of effectively handling multiple requests simultaneously. However, they may present unexpected temperature-related challenges that accelerate aging, causing errors or malfunctions. Moreover, because of process variability, temperature will vary even for identical cores running at the same operating frequency in the processor. In this scenario, we propose *EquiLifeCM*, a framework designed to maximize the lifespan of cloud machines. Given the system's current status and applications' behavior, *EquiLifeCM* automatically allocates workloads across cores from different cloud machines and applies frequency scaling considering core variability.

1 INTRODUCTION

The demand for cloud-based software services has increased the need for solid warehouse infrastructures to support various services like machine learning, biomedical research, and multimedia processing. However, these services have different CPU and memory needs, making it challenging for the server to properly exploit request-level parallelism (RLP), which wisely uses the available resources to execute applications with minimal latency (Lorenzon and Beck Filho, 2019; Navaux et al., 2023). Moreover, transistor scaling and the end of Dennard's Law resulted in increased heat dissipation, directly accelerating the hardware aging process (Shah and Girard, 2020) and impacting the mean time between failures of cloud devices.

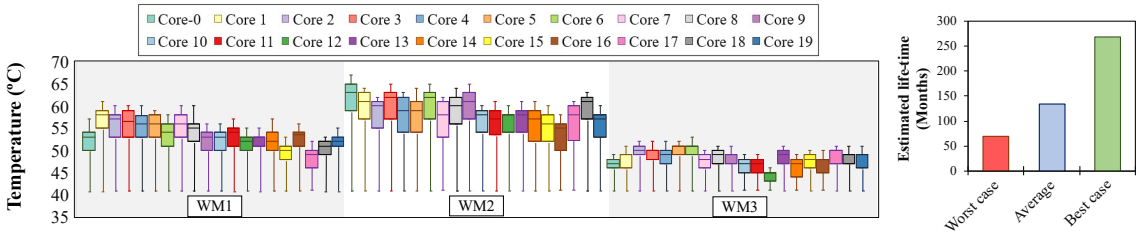
Hardware aging is primarily caused by negative bias temperature instability (NBTI) and hot carrier injection (HCI). NBTI generates positive oxide charge and interface traps in MOS structures under high temperatures and negative gate voltages (Stathis and Zafar, 2006; Blat et al., 1991). This leads to increased threshold voltage (V_{th}), which degrades device performance and can cause undesired system behavior (Schroder and Babcock, 2003). On the other hand, HCI accelerates carriers to high kinetic energies under electric solid fields, resulting in the degradation of oxide quality. This negatively impacts the current drive capability and increases propagation delay. Over time, the accumulation of such damage can cause severe reliability issues (e.g., electromigration, dielectric breakdown, and stress migration (Corbetta and Fornaciari, 2012)), leading to increased maintenance costs and system downtime.

NBTI and HCI are highly influenced by temperature, supply voltage, and operating frequency (Corbetta and Fornaciari, 2012; Medeiros et al., 2020; Medeiros et al., 2021). Therefore, controlling them is essential to reduce aging effects without compromising system performance. Besides the inherent challenge of managing such factors altogether, the within-die process variation makes it even harder: even though off-the-shelf processors have a maximum operating frequency limited by the slowest core's frequency, process variation will influence the temperature of each core, so two identical cores in the same processor running at the same operating frequency may dissipate a different amount of power.

To illustrate this scenario, Figure 1.a shows the temperature distribution of each core in three identical 2x10-core work machines (WM1, WM2, and WM3) that perform the same task under similar conditions in a private cloud setup. The temperature difference across the cores is significant, especially when comparing different non-uniform memory access (NUMA) nodes, represented by cores 0 to 9 and cores 10 to 19. Since temperature impacts aging, this variation will influence the lifespan of each core. Using the aging model described in Section 3, Figure 1.b shows the estimated lifespan of the cores with the highest and lowest peak temperatures and the average lifespan of all cores from Figure 1.a in a variability-oblivious environment. The figure indicates that the

Gonçalves, T., Beck, A. and Lorenzon, A.
Balancing Performance and Aging in Cloud Environments.
DOI: 10.5220/0012687700003711

In *Proceedings of the 14th International Conference on Cloud Computing and Services Science (CLOSER 2024)*, pages 216-223
ISBN: 978-989-758-701-6; ISSN: 2184-5042



a) Temperature Distribution b) Lifetime

Figure 1: (*a*) Distribution of temperature for each core on three identical worker machines when running the same task and (*b*) Estimated lifetime of the worst, average, and best case of cores from (*a*).

worst-case core will last approximately 200 months (16 years) less than the best-case core. Hence, one core can execute 2.58×10^6 more tasks than the other one before reaching its end-of-life.

Based on the previous discussion, we propose ***EquiLifeCM***. It is a new approach to address the aging issue in cloud environments. *EquiLifeCM* considers the impact of process variability on the core temperature and aims to optimize the number of tasks a cloud environment can execute until the machines reach their end. It achieves this by determining (*i*) which worker machine should execute a given task; (*ii*) the assignment of tasks to specific cores; and (*iii*) the frequency and voltage levels for each core and memory system. *EquiLifeCM* consists of two main modules: *Worker-Machines profiling* and *dynamic adaptation*. The profiling module creates a temperature variability map to estimate each core's lifetime in worker machines. It classifies the cores accordingly, storing this data in a database for the dynamic adaptation module. Then, during the task execution, the dynamic adaptation module collects hardware and software events to select the most suitable core automatically and sets optimal core/uncore frequencies to enhance performance and minimize aging.

When executing twenty tasks with different memory and CPU usage behaviors on a private cloud with four identical Intel multicore processors with varying degrees of RLP (25%-100%), we show that: (*i*) *EquiLifeCM* can select the most suitable core to execute a given task among all the available cores in the infrastructure that reduces aging of the hardware components while keeping performance levels of the application as high as possible. (*ii*) Compared to aging-oblivious strategies, *EquiLifeCM* extends server lifetime by four years while enabling the execution of $3.6x10^7$ additional tasks during the lifecycle. (*iii*) *EquiLifeCM* outperforms HiMap (Rathore et al., 2018), a state-of-the-art strategy that maximizes lifetime reliability while meeting performance, power, and thermal constraints.

2 BACKGROUND AND RELATED WORK

Cloud Computing is the go-to technology for application deployment and resource allocation while ensuring elasticity and high availability. However, early cloud systems struggled with compute-intensive applications due to hypervisor overheads. Lightweight container technologies like Docker were introduced to address this. Docker packs applications in containers with all the necessary data, such as libraries, but lacks advanced management features like load balancing. This led to the development of orchestration platforms like Kubernetes, which have become standard in cloud environments (Thurgood and Lennon, 2019). However, Kubernetes does not offer fine-grained optimizations like thread allocation or core/uncore frequency management, limiting its ability to address performance and other specific needs, such as aging.

Multicore architectures can control the operating frequency of the core and uncore subdomains using dynamic voltage and frequency scaling (DVFS) and uncore frequency scaling (UFS), respectively. DVFS allows the software to adjust the clock frequency of a processor's core subdomain in real-time (e.g., processing units and private caches), which reduces power consumption and operating temperature. To make DVFS easier for developers, Operating Systems (OS) provide governors such as *powersave* (reducing power consumption), *performance* (maximizing performance), and *ondemand* (balancing frequency based on CPU load). Similarly, UFS can optimize the frequency of shared elements among cores like the last-level cache, quick-path interconnect controller, and memory controller.

In this scenario, different solutions have been proposed to optimize the execution of applications in cloud environments. Strategies to improve performance and energy efficiency by tuning DVFS and thread mapping include (Dighe et al., 2011; Raghunathan et al., 2013; Stamoulis and Marculescu, 2016;

Schwarzrock et al., 2020; dos Santos Marques et al., 2017). More complete solutions that include aging optimization while meeting performance requirements are discussed next. Hayat is a runtime system that harnesses Dark Silicon to decelerate and/or balance temperature-dependent aging while considering variability to improve the overall system performance for a given lifetime. ADAMAN is an aging-aware task-mapping algorithm that leverages performance, power, and core-level aging predictive models to find energy-efficient mappings that meet the task's performance requirements and reduce platform aging. Lifeguard is an aging enhancement approach based on reinforcement learning that defines the task-to-core mapping based on application performance requirements and the core's safe operating frequency. HiMap is a mapping approach that maximizes the lifetime reliability of multicore systems while satisfying performance, power, and thermal constraints. It determines the mapping of dark cores through a hierarchical approach that finds a cluster of cores to map an application and ensures uniform aging within the cluster.

Our Contributions. Considering the works discussed above, this paper makes the following contributions: (*i*) Compared to strategies that tune the core frequency and thread mapping to optimize applications' performance and energy efficiency, *EquiLifeCM* is a more complete solution as it also considers the uncore subdomain and optimizes aging while not jeopardizing the performance. (*ii*) Compared to strategies that optimize aging under performance requirements, *EquiLifeCM* simultaneously apply task-to-core mapping and core/uncore frequency scaling in a distributed cloud environment to improve the trade-off between performance and aging under process variability. Also, our strategy considers the characteristics of every task during execution to find the most suitable core from all working machines in the environment to execute it.

3 EquiLife CLOUD MANAGER

Our solution, *EquiLife Cloud Manager* (*EquiLifeCM*) is designed to maximize the lifespan of cloud servers while enhancing their performance. It achieves this by uniformly managing the aging of hardware components caused by HCI and NBTI in a cloud cluster by considering the temperature variability between the cores in each NUMA node. This allows *EquiLifeCM* to map tasks to specific cores and control the frequency of both core and uncore subdomains. We illustrate in Figure 2, the workflow of *EquiLifeCM* and discuss it in the following subsections.

3.1 Workflow of *EquiLifeCM*

3.1.1 Initialization

The cloud infrastructure has a set of tasks that need to be executed, regardless of whether they are from the same user (multitenant). These tasks contain datasets that serve as inputs to *EquiLifeCM*. At this stage, *EquiLifeCM* verifies if the variability map for all the working machines is up-to-date and available. If not, the static worker-machine profiling module generates the variability map that contains information about each core's temperature and aging estimation in every architecture from the cloud environment. The variability map must only be generated every few months (configurable in *EquiLifeCM*), as aging is a slow process (Gnad et al., 2015). On the other hand, if the variability map is up-to-date, then *EquiLifeCM* proceeds to the dynamic optimization stage, skipping the stage below.

3.1.2 Static Worker-Machine Profiling

In case there is no variability map (or if it needs to be updated), this stage stresses every core from each working machine to build a new variability map with the core's peak temperature and aging estimation (as discussed next), as shown in Fig. 2. To ensure that all cores are stressed under the same conditions, the stress test only starts when the current core's temperature reaches the idle temperature. The test involves two phases until the peak temperature stabilizes: memory system stress (using the STREAM benchmark) and CPU stress (using the Pi calculation). Throughout the test, *EquiLifeCM* uses OS tools to collect the hardware metrics, including duty cycle, operating frequency and voltage, core temperature, and activity factor. These metrics are used to assess the aging of the corresponding core. After creating the variability map, the cores are classified into two categories based on their estimated aging – the *healthiest* and the *unhealthiest*. These categories are then sorted and used by the dynamic optimization phase to find the best mapping of tasks to the cores. Once this phase finishes, the variability map is sorted through the Merge sort algorithm.

3.1.3 Transparent and Dynamic Optimization

Given the batch of tasks that must be executed in the cloud infrastructure, at this stage, *EquiLifeCM* determines the best way to match tasks with a specific core, as well as the frequency at which core and uncore subdomains should operate based on the task's properties, such as whether it is CPU or memory-intensive).

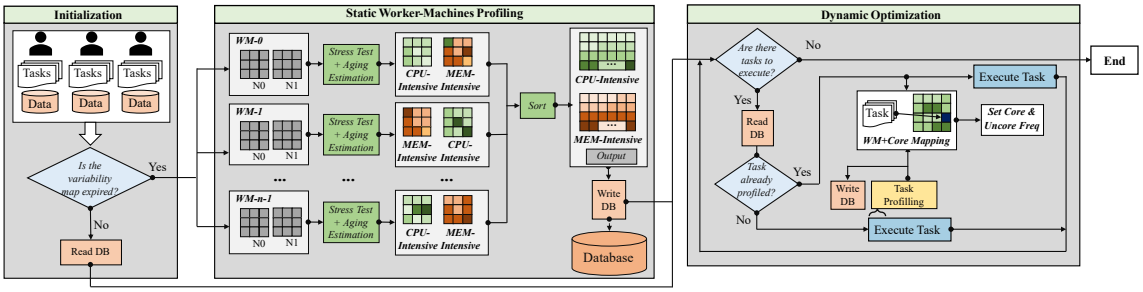


Figure 2: Workflow of *EquiLifeCM*.

For that, *EquiLifeCM* considers that the tasks are encapsulated into Docker containers with all the data and libraries needed to execute them. Then, *EquiLifeCM* first reads the database to get the variability map of each working node previously gathered. From this point on, the task to be executed may pass through dynamic profiling depending on whether it has already been executed.

If it is the first time a task is executed, *EquiLifeCM* performs task-specific dynamic profiling to get hardware/software information from the task. The task is executed during the profiling using the default scheduler configured in the cloud environment (e.g., *Kube-scheduler*), with the core/uncore frequency scaling up or down based on the task's demands. *EquiLifeCM* collects data on the number of instructions per cycle (IPC) and memory usage (such as L2 and last-level cache behavior) to classify the task as either CPU or memory-intensive. We consider these metrics since the IPC determines whether the task is CPU-intensive, while the higher last-level cache access and a low IPC indicate that the task is more memory-intensive. Then, *EquiLifeCM* stores the IPC, L2, and last-level cache behavior on the internal database for future use. If it is not the first time the task is executed, the dynamic profiling is skipped since *EquiLifeCM* recovers the task's characteristics from its database of previous executions.

EquiLifeCM considers that the healthiest NUMA nodes for CPU-intensive tasks are the ones with the lowest peak temperature. Since these tasks require fewer memory operations, the uncore frequency can be set to a minimum, reducing the power consumption and operating temperature without affecting performance. By doing this, the core frequency has more room to be set to the maximum allowed within the TDP limits. On the other hand, for MEM-intensive tasks, the unhealthiest NUMA nodes with the highest peak temperature and aging estimation are better. Since these tasks require more memory operations, the core frequency can scale according to the workload. Reducing power consumption will leave room to increase the uncore subdomain frequency without impacting the core subdomain temperature. This strategy can slow down the aging process of such cores due to lower usage, decreased duty cycle, and lower frequency, which reduces the corresponding voltage and operating temperature.

3.2 Modeling Aging Phenomena

We use a microarchitecture level aging model that considers the combined impact of NBTI and HCI (Lee et al., 2018; Oboril and Tahoori, 2012). NBTI is a phenomenon that negatively affects the electrical properties of pMOS transistors, leading to an increase in the threshold voltage (V_{th}) and decreased switching speed over time. The $|V_{th}|$ shift degree depends on various factors such as supply voltage V_{dd}, duty cycle δ, and usage. HCI primarily affects nMOS transistors and causes $|V_{th}|$ shifts based on the switching activity. In the end, both NBTI and HCI contribute to a gradual increase in MOSFET's threshold voltage (V_{th}), resulting in slower performance.

3.3 Implementation of *EquiLifeCM*

We developed *EquiLifeCM* using the *Python3* programming language in such a way that it can receive a batch of tasks and their datasets in a Docker container without requiring any modifications or recompilation of the task's source code. Additionally, *EquiLifeCM* does not require superuser privileges from the OS, as all of its operations are carried out using tools available at the user level. *EquiLifeCM* uses the `sensors` tool on every working node to read the core temperature and operating voltage, the `taskset` command line to set the task-to-core mapping. The `LIKWID` performance tool[1] to configure the core/uncore frequency. In this scenario, *EquiLifeCM* can work with any task running on cloud environments.

[1]Likwid performance tool available at: https://hpc.fau.de/research/tools/likwid/

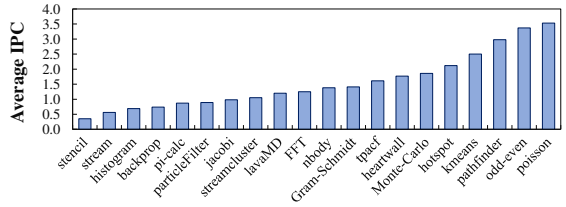


Figure 3: Behavior of each task considering the average IPC.

4 METHODOLOGY

4.1 Applications

We evaluate twenty tasks/applications already implemented in C/C++ from assorted benchmark suites: **Three from the Parboil (Stratton et al., 2012):** simulating histogram (*histo*); 3-D stencil operation (*stencil*); and two-point angular correlation function (*tpacf*).**Eight from the Rodinia Suite** (Che et al., 2009): *Back Propagation*; *Heart Wall*; *HotSpot*; *Kmeans*; *LavaMD2*;*Particle Filter*; *PathFinder*;and *Streamcluster*. **Nine from different domains:** *fast Fourier transform*; *Gram-Schnmidt process*; *Jacobi method*; *Monte-Carlo*; *n-body*; *Odd-even sort*; *Pi calculation*; *Poisson Equation*; and *Stream*.

We executed each task using the standard input set defined on each suite. We chose these applications because they exhibit different behaviors concerning the average IPC, which reflects whether they are CPU- or memory-intensive, as shown in Figure 3. For instance,*Stencil* has the lowest average IPC, meaning that the application spends most of its time on memory operations. On the other hand, *Poisson Equation* has the highest average IPC and is the most CPU-bound application used in the experiments.

4.2 Execution Environment

The experiments were performed on a private cloud with five multicore architectures: four identical worker machines (*W1*, *W2*, *W3*, and *W4*) that execute the tasks; and one *Server*, responsible for deploying the client requests to the worker machines. Each worker machine has the following configuration: 2x 10-core Intel Xeon E5-2650 v3, each core running at a frequency from $1.2GHz$ to $2.3GHz$ ($3.0GHz$ when Turbo Boost is turned on). Moreover, the uncore frequency can scale from $1.2GHz$ to $3.0GHz$. Each task can be scheduled to any of the 80 cores available in the environment. All the worker machines used the Linux Kernel v.4.19.0-21, *kubernetes* v.1.23.6., and *Docker* v.20.10.17, *build* 100c701. We compiled each application with GCC/G++ 12.3 using the $-O3$ optimization flag.

We set up a cluster using the Kubernetes framework, where the master node is responsible for managing the Kubernetes environment and has all the necessary components for governing the cluster, including the Kubernetes control plane components. The management layer in the master node comprises five key components: the *Kube-controller-manager*, which maintains the integrity of the nodes within the cluster; the *etcd* database, which serves as the storage mechanism for all the information related to the cluster; the *Cloud Control Manager*, a component that handles the integration between the cluster and the underlying cloud provider's services; the *Kube-apiserver*, which acts as the front-end interface for the management layer of Kubernetes; and the *Kube-scheduler*, which is responsible for examining newly created pods without assigned nodes and selecting a suitable node for their execution.

The following strategy was used to execute each application on a worker node: (*i*) The application binary is encapsulated within a Docker container to provide an isolated execution regardless of the working node. (*ii*) Based on the decision of *EquiLifeCM*, a Kubernetes Manifest file, referred to as *pod*, is created to describe how the container should run. (*iii*) Using the Kubernetes environment and the *kubectl* command-line tool, the application is deployed for execution through the command `kubectl apply -f file.yaml`, where the last parameter represents the pod configuration file. (*iv*) The Kubernetes control plane creates a pod containing the Docker containers that are responsible for running the application. (*v*) The Kubernetes *kube-scheduler* recognizes the created pod and assigns it to the worker node according to the decision made by *EquiLifeCM*. (*vi*) Finally, the pod executes on the allocated worker node.

5 EVALUATION

5.1 Set of Experiments

We compare *EquiLifeCM* with two state-of-the-art strategies: *Kube-Scheduler*, where the Kubernetes scheduler assigns tasks, and *HiMap* (Rathore et al., 2018), an approach to maximize lifetime reliability while meeting performance, power, and thermal constraints. We have faithfully implemented *HiMap* following the guidelines in the original paper and applied it to the evaluated applications.

Because the workload of cloud servers can vary depending on factors such as the number of users,

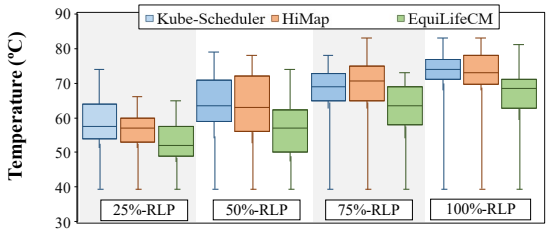


Figure 4: Temperature distribution.

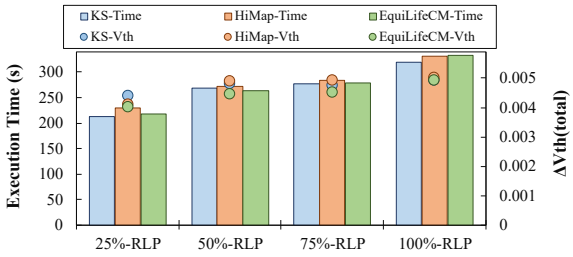


Figure 5: Time and $\Delta V_{th(Total)}$. The lower the value, the better the result.

task type, and service-level agreement, we evaluated all strategies under four scenarios based on the RLP used by the environment. *25%-RLP*, where the batch size matches one-quarter of the total cores (20, for example). *50%-RLP*, the batch size equals half the available cores. *75%-RLP*, the batch size matches three-quarters of the cores. And *100%-RLP*, where the batch size considers all the tasks. Furthermore, to ensure that all experiments were performed with the working machines in the same state (e.g., temperature value), we defined a threshold value of core temperature, and the execution of each scenario only starts when the temperature reaches this value.

5.2 Temperature Evaluation

We start by discussing the temperature behavior of each strategy when executing all scenarios on the private cloud. Figure 4 illustrates the temperature distribution of all working machines in each scenario. Each box includes the minimum and maximum values as whiskers on either side. The three parts inside the box are the first quartile (*Q1*), the median (*Q2*), and the third quartile (*Q3*). Therefore, the higher the values, the higher the operating temperature of the processor. Furthermore, the distance between two quartiles represents 25% of the distribution.

Let us first discuss the behavior of the *KS* policy and *HiMap*. As depicted in Figure 4, the *KS* policy is less effective at controlling temperature as it does not consider core temperature fluctuations in core while assigning tasks. Instead, it prioritizes resource sharing and balance over temperature management. Moreover, the default *KS* task execution mode operates both core and uncore frequency at maximum, adversely affecting temperature behavior. Compared to *KS*, *HiMap* prioritizes map tasks to healthy cores and adjusts the core frequency to optimize the temperature. However, it does not consider uncore frequency management, making a more comprehensive solution necessary.

EquiLifeCM fills this gap by smartly assigning tasks to the most suitable core and adjusting the core/uncore frequency scaling based on the inherent characteristics of each task. *EquiLifeCM* achieves this for two reasons: Firstly, for memory-intensive tasks, it lowers the core frequency/voltage to reduce power consumption without affecting the task's performance. Secondly, for CPU-intensive tasks, *EquiLifeCM* sets the uncore frequency to a minimum allowed level, further decreasing power consumption. These decisions do not impact performance, as demonstrated in Figure 5. Consequently, *EquiLifeCM* reduces peak temperature by up to 13% compared to *KS* (*25%-RLP* scenario) and 12.3% compared to *HiMap* (*75%-RLP* scenario), resulting in a reduction in operating temperature of $9°C$ and $10°C$, respectively.

5.3 Aging and Estimated Lifetime Analysis

In this subsection, we discuss the outcome of *EquiLifeCM* being able to reduce the operating temperature levels of the core while not jeopardizing the application's performance. For that, Figure 5 depicts the execution time for each scenario, represented by bars on the left, and the corresponding change in $V_{th(Total)}$, shown as circles on the right, for each strategy. Similarly, Figure 6 illustrates the number of times a batch of tasks can be executed before the core's end-of-life, represented by bars on the left and the core's expected lifespan, shown as circles on the right. We consider a core at the end of its life when V_th increases by 15%. As an example, if *KS* is used to execute the *25%-RLP* scenario, it can complete around 66M executions before reaching the end-of-life, which is expected to occur in *3.9* years, as represented by the circle on the right.

Applying *EquiLifeCM* offers key advantages. One of the most significant advantages is that it slows down the process of hardware aging and extends the lifespan of cloud worker machines without compromising task performance. The system smartly assigns tasks to the most appropriate cores based on process variability and intrinsic characteristics. This ensures that tasks are completed efficiently and effectively without compromising performance. *Equi-*

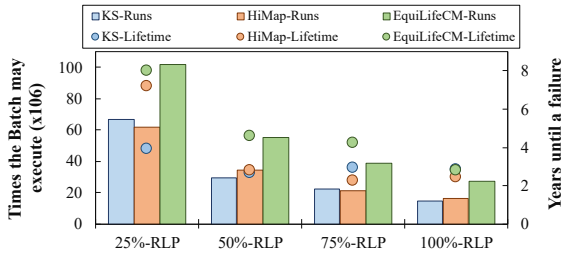


Figure 6: Performance (primary y-axis) and lifetime results for each scenario (secondary y-axis).

LifeCM also manages core and uncore frequency and voltage levels, which helps to reduce temperature without affecting performance.

With that, *EquiLifeCM* achieves a 10% lower $\Delta V_{th(Total)}$ compared to KS (25% RLP) and 9% lower than *HiMap* (50% RLP), maintaining equivalent performance. As RLP increases, so does $\Delta V_{th(Total)}$, affecting temperature. With *EquiLifeCM*, the core's end-of-life is significantly higher, especially at 25% and 50% RLP exploitation, where it can improve processor lifetime by up to 4 and 2 years compared to *KS* and *HiMap*, respectively.

5.4 Overhead of EquiLifeCM

EquiLifeCM incurs overhead in three scenarios: (*i*) At the start of each aging epoch, the *Static Worker-Marchines Profiling* module performs the stress test to update the temperature variability map and estimates core aging, taking about 2.7 minutes per core. This process is required only once every few months (e.g., 3 or 6 months) and has a minimal overhead considering the benefits of *EquiLifeCM*. (*ii*) New tasks are profiled upon execution, with characteristics stored in the database, a step whose benefits outweigh the overhead. (*iii*) A minor overhead related to database management, with *EquiLifeCM* occupying *8.7 Kb*, each task characteristic hash adding *148 bytes*, and database update and search times are minimal at *0.0011s* and *0.0023s*, respectively.

6 CONCLUSION

In this paper, we have presented *EquiLifeCM*, an approach that combines task-to-core mapping with core/uncore frequency management to extend the lifespan of cloud environments. Through evaluating various scenarios across four multicore platforms, we have shown that *EquiLifeCM* can extend the estimated lifetime of cloud machines by 4 and 2 years, outperforming both standard task execution methods and a state-of-the-art strategy. As future work, we aim to develop *EquiLifeCM* further to consider the execution of parallel workloads and heterogeneous architectures.

ACKNOWLEDGEMENTS

This study was financed in part by the CAPES - Finance Code 001, FAPERGS, and CNPq.

REFERENCES

Blat, C., Nicollian, E., and Poindexter, E. (1991). Mechanism of negative-bias-temperature instability. *Journal of Applied Physics*, 69(3):1712–1720.

Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J. W., Lee, S.-H., and Skadron, K. (2009). Rodinia: A benchmark suite for heterogeneous computing. In *IEEE ISWC*, pages 44–54. Ieee.

Corbetta, S. and Fornaciari, W. (2012). Nbti mitigation in microprocessor designs. In *ACM GLSVLSI*, pages 33–38, NY, USA. ACM.

Dighe, S., Vangal, S. R., Aseron, P., Kumar, S., Jacob, T., Bowman, K. A., Howard, J., Tschanz, J., Erraguntla, V., Borkar, N., De, V. K., and Borkar, S. (2011). Within-die variation-aware dynamic-voltage-frequency-scaling with optimal core allocation and thread hopping for the 80-core teraflops processor. *IEEE JSSC*, 46(1):184–193.

dos Santos Marques, W., de Souza, P. S. S., Lorenzon, A. F., Beck, A. C. S., Rutzig, M. B., and Rossi, F. D. (2017). Improving edp in multi-core embedded systems through multidimensional frequency scaling. In *IEEE ISCAS*, pages 1–4. IEEE.

Gnad, D., Shafique, M., Kriebel, F., Rehman, S., Sun, D., and Henkel, J. (2015). Hayat: Harnessing dark silicon and variability for aging deceleration and balancing. In *52nd ACM/EDAC/IEEE DAC*, pages 1–6. IEEE.

Lee, H., Shafique, M., and Al Faruque, M. A. (2018). Aging-aware workload management on embedded gpu under process variation. *IEEE Transactions on Computers*, 67(7):920–933.

Lorenzon, A. F. and Beck Filho, A. C. S. (2019). *Parallel computing hits the power wall: principles, challenges, and a survey of solutions*. Springer Nature.

Medeiros, T. S., Berned, G. P., Navarro, A., Rossi, F. D., Luizelli, M. C., Brandalero, M., Hübner, M., Beck, A. C. S., and Lorenzon, A. F. (2020). Aging-aware parallel execution. *IEEE Embedded Systems Letters*, 13(3):122–125.

Medeiros, T. S., Pereira, L., Rossi, F. D., Luizelli, M. C., Beck, A. C. S., and Lorenzon, A. F. (2021). Mitigating the processor aging through dynamic concurrency throttling. *Journal of Parallel and Distributed Computing*, 156:86–100.

Navaux, P. O. A., Lorenzon, A. F., and da Silva Serpa, M. (2023). Challenges in high-performance computing.

Journal of the Brazilian Computer Society, 29(1):51–62.

Oboril, F. and Tahoori, M. B. (2012). Extratime: Modeling and analysis of wearout due to transistor aging at microarchitecture-level. In *IEEE/IFIP Int. Conf. on Dependable Systems and Networks*, pages 1–12.

Raghunathan, B., Turakhia, Y., Garg, S., and Marculescu, D. (2013). Cherry-picking: Exploiting process variations in dark-silicon homogeneous chip multiprocessors. In *DATE*, pages 39–44.

Rathore, V., Chaturvedi, V., Singh, A. K., Srikanthan, T., Rohith, R., Lam, S.-K., and Shaflque, M. (2018). Himap: A hierarchical mapping approach for enhancing lifetime reliability of dark silicon manycore systems. In *DATE*, pages 991–996.

Schroder, D. K. and Babcock, J. A. (2003). Negative bias temperature instability: Road to cross in deep submicron silicon semiconductor manufacturing. *Journal of applied Physics*, 94(1):1–18.

Schwarzrock, J., de Oliveira, C. C., Ritt, M., Lorenzon, A. F., and Beck, A. C. S. (2020). A runtime and non-intrusive approach to optimize edp by tuning threads and cpu frequency for openmp applications. *IEEE TPDS*, 32(7):1713–1724.

Shah, A. P. and Girard, P. (2020). Impact of aging on soft error susceptibility in cmos circuits. In *2020 IEEE 26th IOLTS*, pages 1–4.

Stamoulis, D. and Marculescu, D. (2016). Can we guarantee performance requirements under workload and process variations? In *ISLPED*, page 308–313, New York, NY, USA. ACM.

Stathis, J. H. and Zafar, S. (2006). The negative bias temperature instability in mos devices: A review. *Microelectronics Reliability*, 46(2-4):270–286.

Stratton, J. A., Rodrigues, C., Sung, I.-J., Obeid, N., Chang, L.-W., Anssari, N., Liu, G. D., and Hwu, W.-m. W. (2012). Parboil: A revised benchmark suite for scientific and commercial throughput computing. *Center for Reliable and HPC*, 127.

Thurgood, B. and Lennon, R. G. (2019). Cloud computing with kubernetes cluster elastic scaling. In *ICFNDS*, NY, USA. ACM.