Postponing Wearout Failures in Chip Multiprocessors Using Thermal Management and Thread Migration

Elham Kashefi, Hamid R. Zarandi
Department of Computer Engineering and Information Technology
Amirkabir University of Technology (Tehran Polytechnic)
Tehran, Iran
kashefi@aut.ac.ir, h_zarandi@aut.ac.ir

Ann Gordon-Ross
Department of Electrical and Computer Engineering
University of Florida, Gainesville, FL-32611
ann@ece.ufl.edu
Also with the NSF Center for High-Performance Reconfigurable Computing (CHREC) at UF

Abstract - This paper presents an improved method to postpone wearout failures and improves functional unit and entire system lifetime by considering two important wearout factors: temperature and functional unit usage. Our method provides a more fine grained approach as compared to prior methods by considering individual functional unit usage. Using this information, system behavior can be predicted and appropriate thread scheduling and migration decisions can be made. Our method incorporates temperature predictions based on recent historical temperatures and functional unit usages to rank threads and cores in a chip multiprocessor (CMP). Using these rankings, our method migrates threads among cores to reduce thermal hotspots. Simulation results on the ESESC simulator show that our method can improve the average system temperature and lifetime by approximately 4.33°C and 21.65%,respectively,in a tri-core CMP, and 6.4°C and 32% in a quad-core CMP.

***Keywords**: Wearout; Temperature; Chip multiprocessor; Thread migration*

I. INTRODUCTION

In new technology generations, inappropriate voltage scaling has intensified current and power densities, which results in rapidly increasing chip temperature [1]. Wearout mechanisms are intensified by temperature, power, and system activity, thus wearout faults will become increasingly prevalent in future technology generations [1].Concomitantly, shrinking CMOS feature size accelerates transistor wearout, and consequently processor lifetime is becoming more unpredictable and shorter than expected [1][2]. For example, a delay increase of just 0.04*ps* per logic cell within an ALU can lead to more than a 20*ps* increase in the delay, or about one inverter delay, on the ALU result bus [1]. This exemplifies how wearout latency at the device level is magnified much more at the architectural level.

Even though the time to wearout failure heavily depends on factors, such as technology, manufacturing process, temperature, voltage conditions, etc., wearout is also impacted by factors, such as usage frequency (duty cycle), which can be moderated/balanced to increase chip lifetime. The time duration in which hardware is being used plays an essential role in the onset of wearout. As the hardware usage increases (i.e., the hardware is stressed), wearout causes are intensified and manifested sooner. Thus, one method to postpone wearout failures is to reduce hardware stress [3]. Since even small changes in usage and/or temperature has a significant effect on estimated aging, it is important to consider both of these factors at the same time.

Several wearout recovery and wearout mitigation techniques in previous works have tried to balance the number of executed instructions (i.e., workload/stress) over multiple functional units to increase the units' lifetimes [4][5][6][7]. However, prior work has shown that balancing the number of executed instructions and workloads among functional units is not always the best strategy for mitigating aging effects [8].Different instructions which use the same functional unit, have different delays and clock cycle slack times. This difference is essential when considering aging (not performance). *Critical* instructions have very little slack time. When path delay increases due to aging effects, critical instructions start to fail due to this increased delay, however, non-critical instructions can tolerate more delay, and execute correctly for a longer period before suffering wearout effects [8].

Thus, since different instructions place different usage stresses on different functional units, these factors should be considered since the first functional unit to fail ultimately dictates total system lifetime. Based on this motivation, we propose a method that considers essential factors that accelerate wearout—temperature and usage time—of the functional units, predicts system behavior, and attempts to postpone wearout failures. The main contributions of this work are:

1) *Thermal management thread scheduling decisions based on fine-grained wearout factors:* Functional units in cores are considered individually, and one decision does not apply to all of the threads on that core, which improves the accuracy of thermal management during thread scheduling.

2) *Temperature prediction based on the rate of change of two historical data set factors (temperature and usage):* Predicting temperatures based on two different data sets can improve the prediction accuracy.

3) *Temperature compatibility with application behavior*: There is a close relationship between temperature and an application's behavior, and

migrating threads to different cores provides compatibility with the thermal demands of a thread, which can improve system performance.

Simulation results on the ESESC simulator show that our method can improve the average system temperature and lifetime by approximately 4.33°C and 21.65%, respectively, on a tri-core chip multiprocessor (CMP), and 6.4°C and 32%, respectively, on a quad-core CMP
The rest of this paper is organized as follows. Section II gives an overview of relative prior work. In Section III, our proposed method is presented. Experimental setup and evaluation for a set of SPEC2000 benchmarks are given in Section IV. Finally, we conclude our work in Section V.

II. RELATED WORK

Some architectural level methods [4][5] consider instructions per cycle (IPC) as a factor for task scheduling, in which workloads with higher IPC are assigned to cores with lower temperature. Oboril et al. [8] classifies instructions according to their criticality. Critical instructions are executed on different parts of the processor (units) that use special mitigation techniques to increase the units' lifetimes. All other non-critical instructions are executed on units without these mitigation techniques. Siddiqua et al. [6] varies the functional unit scheduling to try to balance wearout across all units. Priority rotation scheduling and time-dependent scheduling are proposed, which create a recovery period after a stress period for a functional unit. Sun et al. [3] also proposes that cores or units enter a recovery period after a stress period. Khan et al. [9] proposes a scheduling method for temperature and age balancing, where applications are ranked according to the application's current temperatures and cores are ranked based on the core's frequency degradation. Sunet al. [7] categorizes cores into zones, and uses dynamic task scheduling to balance the workload across all zones. Each core has a defined metric indicating how much that core can be stressed by additional workloads.

Some architectural level methods consider past circumstances of a core using statistical methods and unit workloads to balance future workloads or shut down over stressed cores[3][6][7][9]. However, these methods consider all units together, and per-unit workload is not considered. Our proposed method improves upon prior methods by considering all units separately, which provides a more fine-grained evaluation of overall chip wearout.

III. OUR PROPOSED METHOD

Our proposed method uses two per-unit historical input data factors—temperature and usage time—to predict the future data values for a thread running on a core. Based on these predictions, threads can be scheduled to the most suitable core to balance wearout, and thread migration decisions can also be made. Figure 1 shows a high-level

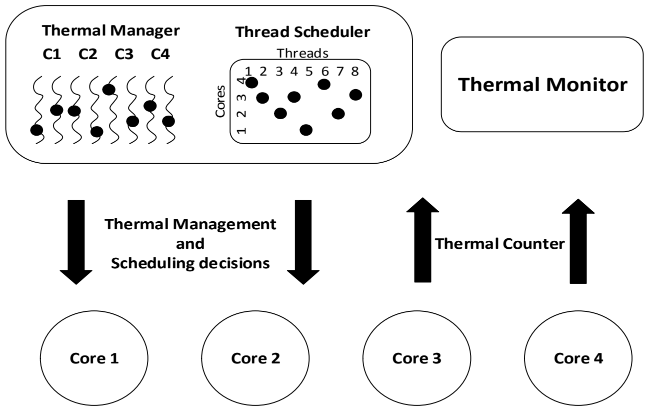


Figure 1. A high level view of proposed method

view of system operation. At runtime, two controlling units monitor the cores thermal activity and the threads' core assignments: a thermal manager and a thread scheduler. The thermal manager records the temperature of the cores' functional units. The thread scheduler uses this thermal information along with usage history, and considering processor performance, schedules and migrates threads to avoid core hotspots.

A time quantum comprises of two types of time segments: one short segment and one long segment. The main purpose of the short segment is to sample the temperature and usage of the functional units over a short time period. Periodic samples are taken during the short segment, and at the end of the short segment, the thread scheduler uses tis historical data to determine thread-to-core scheduling decisions and migrations. Subsequently, the long segment runs the thread up to the end of the time quantum.

Based on experimental results, a short segment comprises 40,000 instructions. At the end of the short segment, the thermal manager evaluates the per-thread data which indicates the threads' functional unit requirements, along with each cores' thermal and functional unit capabilities. This information, along with the estimated thread migration overhead is used to determine if a thread should be migrated when hotspots are predicted based on the recorded historical sampled data. When a thread's execution ends, or a new thread is generated, or a quantum ends, a new short segment begins.

The thread migration process uses three main data structures: the threads-to-cores mapping structure, the historical temperatures structure, and the core reliability structure. The threads-to-cores mapping structure records the current thread-to-core mappings for subsequent migration decisions. The historical temperatures structure maintains the historical temperature values for each thread running in the system. The core reliability structure maintains the aging state of each cores' functional units. All of these structures' data are used in thread scheduling and migration decisions to ensure that threads are mapped to the most suitable cores based on the cores' thermal conditions. Historical samples are maintained for the short segment, and

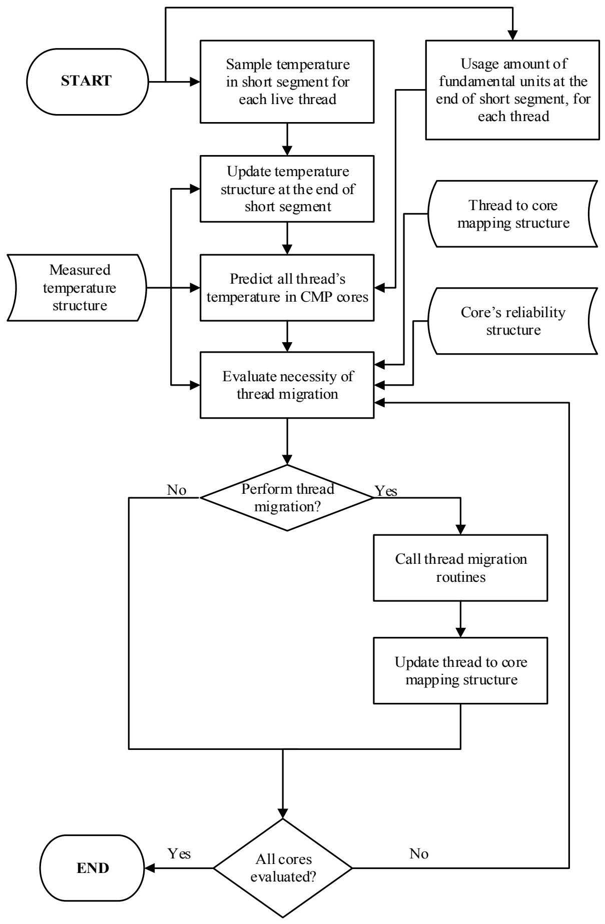

START
Sample temperature in short segment for each live thread
Usage amount of fundamental units at the end of short segment, for each thread
Update temperature structure at the end of short segment
Thread to core mapping structure
Measured temperature structure
Predict all thread's temperature in CMP cores
Core's reliability structure
Evaluate necessity of thread migration
No
Perform thread migration?
Yes
Call thread migration routines
Update thread to core mapping structure
END
Yes
All cores evaluated?
No

Figure 2. Flowchart of the proposed method

each new time quantum replaces the old data.

Figure 2 depicts a flowchart of our proposed method. During the short segment, the thermal manager periodically records the running threads' core's temperatures in the historical temperatures structure, and determines if thread migration should occur at the end of the short segment. Thread migration decisions are based on the predicted future thermal conditions for each core and the cores' estimated remaining time until chip hotspot threshold violations. Based on this estimated time, a thermal and usage ranking of threads, highest to lowest, is determined based on the estimated shortest time to a predefined temperature threshold violation. After the threads are ranked, the threads are mapped to suitable cores. Based on a thread's rate of change (*ROC*) in historical temperature, the approximate remaining time before that thread's temperature threshold is exceeded can be estimated as [10]:

$$Temp + ROC \times X = Target\_Temp \qquad (1)$$

where ROC is the temperature rate of change (calculated using the historical samples), $Temp$ is the current temperature, $Target\_Temp$ is predefined temperature threshold, and X is an unknown variable that indicates the

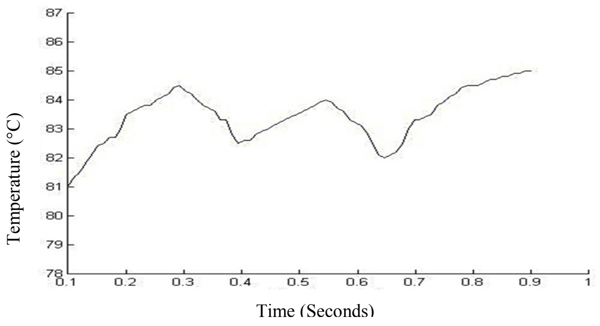


Figure 3. Thermal behavior for the *fpALU* unit while running *bzip2*

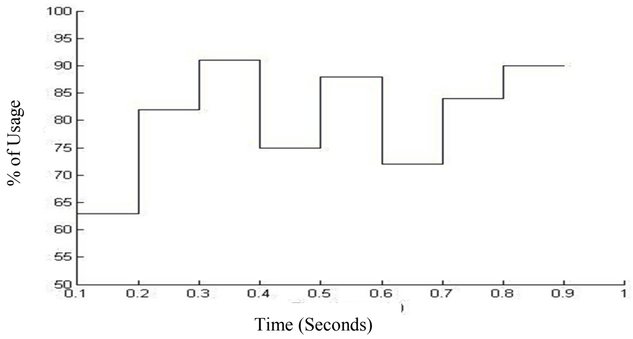


Figure 4. Usage time for the *fpALU* unit while running of *bzip2*

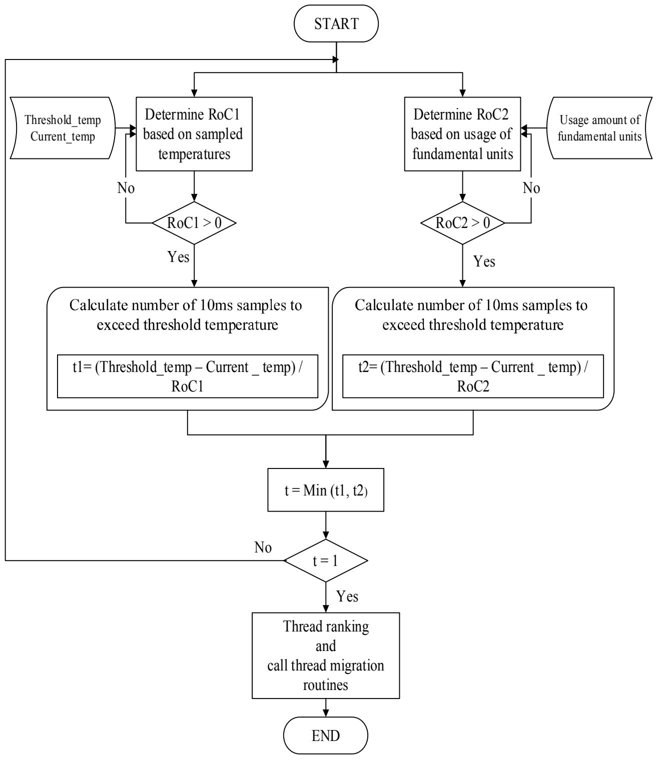
remaining time before the temperature is expected to exceed $Target\_Temp$. Similarly to [10], sampling is done on a $100us$period. Since the ROC is generally non-linear, averaging historical temperature samples is not accurate for calculating ROC, thus the temperature samples are weighted based on age, where newer samples are weighted more heavily. We empirically evaluated different numbers of samples and several weighting schemes to determine the best combination for calculating ROC. Experiments showed that using more than eight samples incurred high performance overhead (samples are gathered using several instructions), using less than four samples resulted in inaccurate predictions, and using six samples provided a good performance to prediction accuracy tradeoff. Experimenting with different historical weightings showed that when using six samples, weights of 60%, 50%, 40%, 30%, 20%, and 10% for the newest to oldest samples, respectively, provided the best correlation with actual execution results.

The functional units' usage times are considered as an important factor when ranking threads. In previous methods, IPC [4][5] has been used as a factor for measuring aging and for ranking threads and cores, however, these methods did not take into account which actual functional units were used and for how long. Prior work showed that running a benchmark does not increase the functional units' temperatures equally, and the temperature distribution in all units is not monotonic [11]. A functional unit's temperature increase highly depends on how long an executing instruction uses each functional unit, and all instructions do not use the same percentage of the clock cycle. To show this correlation between temperature and percentage of usage time, Figure 3 depicts the thermal behavior over time for the floating point ALU unit (*fpALU*) while running the *bzip2* benchmark and Figure 4 depicts the usage percentage of this unit at the same time. As expected, trend similarities are obvious when comparing the two diagrams—a unit's temperature is directly related to the unit's usage percentage.

Given this strong correlation, our method considers the functional units' usage times when predicting a core's temperature while executing a particular thread. Prior work [12] classified some instructions as *functional* instructions, which are instructions with high rates of occurrence, such as load, store, branch, integer ALU operations, and floating point ALU operations. Since these are the most common instructions, based on Amdal's law, we need only consider the different functional unit usages for these instructions, and can safely disregard the other instructions. The functional units used by these instructions are: *Load*, *Branch*, *Store*, *iALU*, and *fpALU*, respectively, where *i* and *fp* imply integer and floating point, respectively. Therefore, these functional units' usage times are included when evaluating and predicting a core's thermal behavior given a threads instruction mix, and considering each different functional unit separately during thermal management improves the prediction accuracy.

Therefore, a usage ranking of the threads is determined using the thread's execution behavior and functional unit usage, which is derived from the number of committed *functional* instructions during a short segment. This information, along with the cores' functional units' collective aging time, can be used to balance aging. A per-core table records the historical execution frequency for each of the five functional instructions. These frequency values can be tabulated using a counter to count the number of times each of the functional instructions commit, and can be integrated into the core with a few changes to the instruction commit unit. At the end of each sampling period, the counter for the functional instruction type in that segment dictates the usage percent of the instruction's related functional units, and is used for making thread scheduling and migration decisions.

At the end of a short segment, Figure 5 depicts the thermal manager's flowchart for each thread when a core's temperature's rate of change or a functional unit's usage rate of change increases, which signals that thread migration decisions should be made. In this flowchart, decisions are made based on the rate of change of the cores' temperatures and the cores' functional units' usage. If the rate of change is positive, the associated core's expected time to exceed the threshold temperature is predicted using Equation (1), and this time is used to rank the cores highest to lowest—shortest time to longest time—and the highest ranked core's thread is selected for migration. Based on this ranked order, the threads are migrated to other cores, beginning with the most aged core (minimum time remaining to exceed the thermal threshold). Core aging is estimated using the five functional units, which are ranked based on the unit's aging and temperature, and using these rankings, the cores can be ranked by age. Next, thread migration is done if necessary. To avoid frequent thread migrations (i.e., migration thrashing), we use a 0.5°C temperature threshold (based on [10]) as the temperature difference between two cores to initiate a thread migration. The following thread migrations only occur when this threshold is exceeded. The thread with the highest rank is migrated to the most aged core, the second lowest ranking thread is migrated to the second most aged core, and so on until all threads are assigned to appropriate cores.


START
Threshold_temp
Current_temp
Determine RoC1 based on sampled temperatures
Determine RoC2 based on usage of fundamental units
Usage amount of fundamental units
No
No
RoC1 > 0
RoC2 > 0
Yes
Yes
Calculate number of 10ms samples to exceed threshold temperature
t1= (Threshold_temp – Current _ temp) / RoC1
Calculate number of 10ms samples to exceed threshold temperature
t2= (Threshold_temp – Current _ temp) / RoC2
t = Min (t1, t2)
No
t = 1
Yes
Thread ranking and call thread migration routines
END
Figure 5. Thread Migration Flowchart

IV. EXPERIMENTAL SETUP

For our experiments, we used ESESC [13], which is a cycle accurate architectural multiprocessor simulator with detailed power, thermal, and performance models for modern out-of-order multicores. We modified ESECS to include our proposed method. Table 1 lists the major architectural parameters. We used the SPEC2000 benchmark suite, and grouped the benchmarks to create multithreaded workloads as depicted in Table 2. Each workload is a combination of floating point and integer type benchmarks. The benchmarks are selected based on their thermal characteristics and instruction types (integer and floating point) [12] such that each workload is a combination of different benchmarks with different characteristics in terms of processor utilization and temperature. We used a thermal threshold value of 85°C

Table 1. Core Architectural Parameters

Core Parameters	
Branch Predictor	8K Hybrid 2-level
Branch Target Buffer	4K entries, 4-way
Front-End Width	4-way
Fetch Queue Size	32 entries
Load/Store Buffers	64,48
Retire Width	6 (out-of-order)
Integer Issue Queue	48 entries, 4-way issue
Integer Register File	80 registers
Integer Execution Units	iALU: 5 units, 1 cycle
FP Issue Queue	24 entries, 1-way issue
FP Register File	80 registers
FP Execution Units	fpALU: 2 units, 2 cycles
Memory Hierarchy	
L1 Instruction Cache	64KB, 8-way associative, 1 cycle, LRU
Load Queue	64 entries
Store Queue	48 entries
L1 Data Cache	64KB, 8-way associative, 2 cycles, LRU, WB
Last Level Cache	2MB, 8-way associative, 10 cycles, LRU
Main Memory	200 cycle latency (100 ns @ 2 GHz)

Table 2. Workloads elected in Experimental Study

Workload	SPEC2000 group	Floating Point/ Integer
WL1	*ammp, art, equake*	FP, FP, FP
WL2	*art, ammp, gcc*	FP, FP, INT
WL3	*ammp, bzip2, gcc*	FP, INT, INT
WL4	*art, ammp, equake, vpr*	FP, FP, FP, INT
WL5	*ammp, equake, twolf, bzip2*	FP, FP, INT, INT
WL6	*art, bzip2, ammp, twolf*	FP, INT, FP, INT

Table 3. Core Degradation Due to Hard Faults [15]

degradation	Assumed list
Increased Leakage	None
	2X nominal in L1 cache and TLBs
	2X nominal in front-end and ROB
	2X nominal in integer back-end
	2X nominal in floating point back-end
	2X nominal in load and store queues
	2X nominal across core (excluding L2)

similarly to [14]. Prior work[15], focus on visible effects of variations and wearout over the lifetime of the system, and focus on three forms of processor degradation:1) modeling the errors that disable one unit/core; 2) core frequency degradation from manufacturing process variation; and 3) excessive leakage currents that lead to very serious problems. Table 3 shows the degraded CMP configurations that are considered in our experiments. As the leakage current increases, the chip temperature also increases, which is representative of aging [16]. Thus, we consider leakage variations that affect an entire core (e.g., [16]), as well as those that affect one or more functional unit.

Figure 6 evaluates a tri-core system for the *fpALU* unit utilization with and without using thermal management, to evaluate a single unit impact on core aging. Table 3 depicts the different degradation rates assumed in the cores' *fpALU* units. The leakage current, and consequently the degradation of the *fpALU* in the first core, represents the fastest aging. The second core's aging is assumed to be less than the first core, and the third core is assumed to have no aging effects.

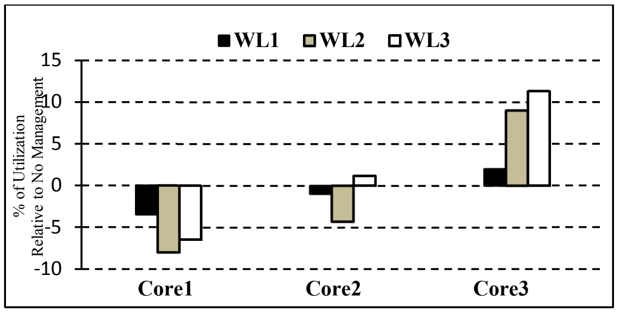


Figure 6. Utilization change of *fpALU* relative to no thermal management in a tri-core CMP

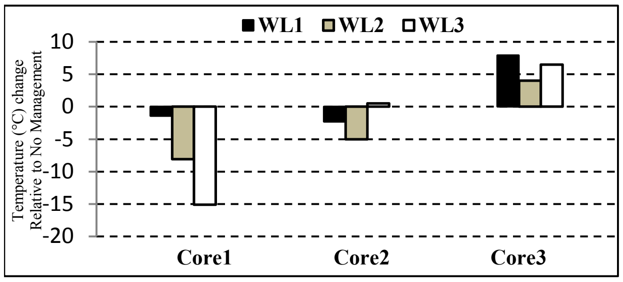


Figure 7.Temperature change of fpALU relative to no management in a tri-core CMP

Three workloads, WL1, WL2 and WL3 as shown in Table 2, are considered in this experiment, which provides varying integer and floating point benchmark combinations for a generalized evaluation. Each workload has at least one floating point benchmark, and our selected benchmarks have higher average temperatures as compared to the other SPEC2000 benchmarks. In Figure 6, utilization is defined [12] as the time in which the corresponding functional unit (*fpALU*) is active, relative to the total time of the workload execution. Because WL1is composed of three floating-point-intensive benchmarks, the usage degradation of the *fpALU* unit in the first core does not changed significantly, while WL2 and WL3 have better solutions.

Figure 7 shows the temperature changes using the measured average temperatures for running threads for the *fpALU* unit, with and without our thermal management. The average temperature of WL1 for the degraded *fpALU* on the first core decreases by approximately 1.4 degrees, where the utilization of this unit in Figure 6 has not decreased. Based on figures 6 and 7, there is significant improvement for the second and third cores while executing WL1.

The reason for the results in figures 6 and 7 for WL1 is because WL1's benchmarks use a large percentage of floating point ALU instructions, which use the *fpALU* unit. Based on this condition, the appropriate thread-to-core mapping could not be created. On the contrary, WL2 and WL3 show much better results. For WL2, the temperature decreases by 8.1 degrees, and the utilization of the aged core

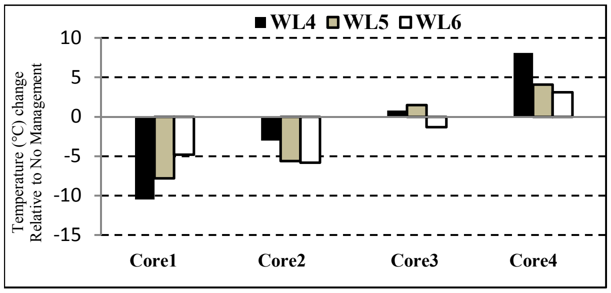


Figure 8. Temperature change of *fpALU* relative to no management in a quad-core CMP

Table 4. CMP Throughput Relative to No Thermal Management

Tri-core CMP throughput	**WL1**	**WL2**	**WL3**
	71%	78%	85%
Quad-core CMP throughput	**WL4**	**WL5**	**WL6**
	80%	94%	88%

decreases as well since the *fpALU* unit is utilized less. For WL3, the first core's temperature reduces by 15.1 degrees and the utilization of the aged core reduces by 6.4%.

In these three workloads, the average temperature of the *fpALU* unit in the first core decreases by 8.2 degrees, the second core has less aging, and the average temperature decreases by 2.26 degrees, and the average temperature for the third core increases by 6.13 degrees. Overall for these experiments on a tri-core system, the average system lifetime is improved by 21.65% for all cores. These results represent the worst-case conditions using highly floating-point-intensive benchmarks and worst-case experimental conditions. Workloads containing fewer floating-point-intensive benchmarks and/or more integer-intensive benchmarks, and more favorable experimental conditions would reveal better results.

Figure 8 depicts similar experiments for a quad-core processor. In these experiments, we use the WL4, WL5 and WL6 workloads. With these workloads, the average temperature of the *fpALU* unit is reduced by 7.7 degrees, 4.8 degrees, 1 degree, and 5.1 degrees, in the first, second, third, and fourth cores, respectively. Based on the achieved results, the overall system lifetime for these experiments on a quad-core processor is improved by 32% on average.
We evaluate performance using CMP throughput [12] defined throughput as the total number of committed instructions in a thread divided by the execution time of the thread. The evaluated CMP throughput is the sum of the throughput of all of the threads [12].

In our experiments, each workload is executed on a processor in two methods: with and without our proposed thermal management. Then, for each workload, the relative CMP throughput is calculated for both methods. The results of these experiments are depicted in Table 4. When considering the experimental conditions and the simulated aging of the *fpALU* unit, when appropriate thread scheduling and migration cannot occur, threads may be relocated several times, which could be the cause of more throughput overhead in some cases in Table 4.

V. CONCLUSION

Due to intensive CMOS technology scaling, wearout mechanisms become a major reliability challenge, increasing a digital circuit's delay, thus shortening the chip's lifetime. In this paper, we mitigate wearout effects by considering two prominent wearout factors: temperature and per-functional unit usage time. Using historical data of these factors, more accurate thermal predictions can be made, and more appropriate thread-to-core mappings and thread migrations can be determined. Our experiments for the SPEC2000 benchmark suite show that the proposed method can improve system lifetime by 21.65% on average on a tri-core CMP, and 32% on average on a quad-core CMP.

VI. ACKNOWLEDGEMENTS

The authors gratefully acknowledge the scientific support and assistance from members of Design and Analysis of Dependable Systems (DADS) laboratory at Amirkabir University of Technology, the laboratory where this work is done. We are thankful for all members who helped this work to be prepared and presented very well.

Moreover, this work was financially supported by the National Science Foundation (CNS-0953447). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] J. A. Blome, et al., "Online timing analysis for wearout detection," *in Workshop on architectural reliability*, 2006.
[2] R. Balasubramanian and K. Sankaralingam, "Virtually-aged sampling DMR: unifying circuit failure prediction and circuit failure detection," *46th Annual IEEE/ACM International Symposium on Microarchitecture*, 2013.
[3] J. Sun, et al., "NBTI aware workload balancing in multi-core systems," *in Quality of Electronic Design*, 2009.
[4] J. A. Winter, D. H. Albonesi and C. A. Shoemaker, "Scalable thread scheduling and global power management for heterogeneous many-core architectures," *19th international conference on Parallel architectures and compilation techniques*, 2010.
[5] C. Zhu,et al., "Three-dimensional chip-multiprocessor run-time thermal management," *IEEE Transactions onComputer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 8, pp. 1479-1492, 2008.
[6] T. Siddiqua and S. Gurumurthi, "A multi-level approach to reduce the impact of NBTI on processor functional units," *20th Symposium on Great leakages symposium on VLSI*,2010.
[7] J. Sun, et al., "Workload assignment considering NBTI degradation in multicore systems," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 10, no. 1, pp. 4, 2014.
[8] F. Oboril, et al., "Reducing NBTI-induced processor wearout by exploiting the timing slack of instructions," *eighth*

IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, 2012.

[9] H. Tajik, H. Homayoun and N. Dutt, "VAWOM: temperature and process variation aware wearout management in 3D multicore architecture," *50th AnnualDesign Automation Conference (DAC)*,2013.

[10] O. Khan and S. Kundu, "Microvisor: A runtime architecture for thermal management in chip multiprocessors," *T. HiPEAC,* vol. 4, pp. 84-110, 2011.

[11] C. Chen and L. Milor, "System-level modeling and microprocessor reliability analysis for backend wearout mechanisms," *Design, Automation & Test in Europe Conference & Exhibition (DATE),* 2013.

[12] O. Khan and S. Kundu, "Thread relocation: a runtime architecture for tolerating hard errors in chip multiprocessors," *IEEE Transactions onComputers*, vol. 59, no. 5, pp. 651-665, 2010.

[13] E. K. Ardestani and J. Renau, "ESESC: A fast multicore simulator using time-based sampling," *High Performance Computer Architecture (HPCA)*, 2013.

[14] R. Z. Ayoub and T. S. Rosing, "Predict and act: dynamic thermal management for multi-core processors," *2009 ACM/IEEE International Symposium on Low Power Electronics and Design*, 2009.

[15] J. A. Winter and D. H. Albonesi, "Scheduling algorithms for unpredictably heterogeneous cmp architectures," *Dependable Systems and Networks (DSN),* 2008.

[16] M. Agarwal, et al., "Circuit failure prediction and its application to transistor aging," *VLSI Test Symposium*, 2007.