Topology-Aware Reliability Optimization for Multiprocessor Systems

Jie Meng[†], Fulya Kaplan[†], Mingyu Hsieh[*], and Ayse K. Coskun[†]
[†]Electrical and Computer Engineering Department, Boston University, Boston, MA – {jiemeng, fkaplan3, acoskun}@bu.edu
[*]Sandia National Labs, P.O.Box 5800, Albuquerque, NM – myhsieh@sandia.gov

Abstract—**High on-chip temperatures adversely affect the reliability of processors, and reliability has become a serious concern as high performance computing moves towards exascale. While dynamic thermal management techniques can effectively constrain the chip temperature, most prior work has focused on temperature and reliability optimization of a single processor. In this work, we propose a topology-aware workload allocation policy to optimize the reliability of multi-chip multicore systems at runtime. Our results show that the proposed policy improves the system reliability by up to 123.3% compared to existing temperature balancing policies when systems have medium to high utilization. We also demonstrate that the policy is scalable to larger systems and its performance overhead is minimal.**

I. Introduction

As the number of cores and power density per processor increase, reliability is becoming a significant concern in high performance systems. High temperatures jeopardize the reliability of the chips and significantly impact performance, while increasing the cooling costs.

In modern processors, temperature and reliability challenges are addressed by management techniques such as clock-gating and dynamic voltage-frequency scaling. Recent methods rely on OS-assisted workload scheduling to regulate chip temperature at reduced performance cost [1, 2, 3, 4, 5]. The main idea behind thermally-aware workload allocation is to exploit temperature variations resulting from executing jobs with different CPU usage profiles. "Hot" jobs, such as computation-intensive algorithms, cause the chip to run at a higher temperature than "cool" jobs, for which most work involves data transfers between memory and processor. Through intelligent scheduling of such hot and cool jobs, we can reduce thermal hot spots and variations. Such temperature-aware workload management approaches have been proposed for both single-core [6, 7] and multicore processors [1, 2, 4, 5].

Among temperature-aware workload management policies, temperature balancing has been shown to be effective at the processor level (e.g., [1]). However, reliability benefits of thermal balancing for systems with multiple nodes (or chips) have not been clear. In this work, we first demonstrate that for systems with multiple chips, *clustering* jobs with higher power consumption can result in higher system reliability compared to aggressively *balancing* the temperature. The reason for this potential benefit lies in the inherent parallelism in multi-chip systems, where failure of a node does not cause the failure of the entire system. Therefore, instead of equally stressing all nodes, clustering higher temperature loads together and maintaining some of the nodes at lower thermal stress levels can extend overall system reliability. This observation is particularly interesting considering many future high-performance systems will be built using multiple multicore chips.

Following an analysis of the tradeoffs between balancing and clustering, we propose a novel policy to optimize system reliability at runtime. Our policy is aware of the parallelism provided by the system *topology* and selects among workload *clustering* and *balancing* approaches to maximize system reliability, while adhering to temperature thresholds as well as cooling and performance constraints. Our specific contributions are as follows:

- Using a detailed reliability modeling approach to accurately model temperature-induced wear-out failure mechanisms and various system topologies, we analyze the reliability of a real-life multi-chip multicore system. Our analysis quantifies the tradeoffs between *clustering* higher power jobs and *thermal balancing* at various operating temperatures. We show that clustering can improve system reliability by up to $4.85X$ for systems with a processor-level parallel topology and 80^oC peak temperature.
- We introduce a topology-aware job allocation policy to optimize the system reliability, targeting systems with medium to high utilization (e.g., as in high-performance clusters). We design low-cost predictors to estimate application power and chip peak temperature during allocation. Our policy adapts to workload changes while respecting thermal constraints.
- We provide an experimental validation using a large set of workload mixes representing different utilization levels and CPU usage profiles. Our policy improves the system reliability by up to 123.3% compared to temperature balancing policies. We also demonstrate the scalability of the proposed policy to larger systems.

II. Related Work

A number of approaches on reliability management focus on microarchitectural optimization [8, 9]. Recent work has also introduced reliability management techniques specifically targeting multicore systems. Hanumaiah et al. [10] optimize the reliability of a multicore processor running tasks with hard deadline constraints by solving a quasiconvex optimization problem. Wang et al. maximize the lifetime of multicore systems while maintaining a given aggregate processor speed by applying sequential quadratic programming [11]. Coskun et al. propose a simulation framework to evaluate the impact of management policies on processor lifetime and demonstrate benefits of temperature balancing [1].

Several reliability management techniques consider both the wear-out mechanism and the system topology. Huang et al. [12] use the Weibull distribution to model aging effects. *RAMP* uses Monte Carlo simulations and lognormal distributions to compute reliability, and a simple MIN-MAX approach to model series-parallel topologies [13].

Recent research has also introduced temperature-aware job allocation policies. Moore et al. develop a temperature-aware workload placement algorithm through establishing a prioritized list of servers for saving energy in data centers [14]. Coskun et al. design adaptive scheduling policies that leverage thermal sensor readings for reducing temporal and spatial temperature variations [15]. Wang et al. propose a thermally-aware job scheduling algorithm for data centers to allocate workloads based on their task-temperature profiles [16].

Our work differentiates from prior research as we focus on the impact of system topology on reliability. Following our analysis that shows *clustering* may provide better reliability than *balancing* depending on the topology, we propose a job allocation method to optimize reliability for multi-chip multicore systems.

III. Reliability Modeling Methodology

In this work, we consider three major intrinsic wear-out failure mechanisms for processors: *Electromigration (EM)*, *Time Dependent Dielectric Breakdown (TDDB)*, and *Negative Bias Temperature Instability (NBTI)* [13, 17, 18]. Failure rates for these three failure mechanisms can be expressed in the following general form:

$$\lambda = \lambda^0 \times e^{\frac{-E_a}{kT}} \tag{1}$$

where E_a is the activation energy for the failure mechanism, k is the Boltzmann's constant ($8.62 \cdot 10^5$), T is the temperature, and λ^0 is a material-dependent constant. $E_{a_{EM}} = 0.7eV$ for Al alloys [17]. We set $E_{a_{TDDB}} = 0.75eV$ [17]. NBTI activation energy represents $E_{a_{NBTI}} \times 1/n$, where n is the measured time exponent. We use $E_{a_{NBTI}} = 0.15eV$ and $n = 0.25$, giving the product $0.6eV$ [17, 18].

To determine the constants for λ^0_{EM}, λ^0_{TDDB}, and λ^0_{NBTI}, we assume the contributions of EM, TDDB, and NBTI are similar to each other at a base temperature. We calibrate the constants in each failure rate equation to satisfy a per-core MTTF of 5 years at 60^oC [19].

A. Lognormal Distributions for Lifetime Reliability

Recent work has shown that lognormal distribution constitutes a more accurate model of wear-out failure mechanisms compared to exponential distribution [13, 20]. Lognormal distribution provides the ability to model the dependence of the failure mechanisms on time. The probability density function for lognormal distribution is given by:

$$f(t) = \frac{1}{t\sigma\sqrt{2\pi}} e^{-\frac{(\ln t - \mu)^2}{2\sigma^2}} \tag{2}$$

where μ and σ are the mean and the standard deviation of the underlying normal distribution, respectively. Reliability at time t can be computed by integrating f(t) from 0 to t. We use $\sigma = 0.5$ based on experimental data from prior work [13].

In order to obtain the reliability of a processor at a certain time, we need to calculate the reliability of each wear-out failure mechanism using lognormal distribution. However, since there is no closed-form solution for the integration of f(t), it is difficult to find an explicit solution for the failure rate or reliability. To address this issue, we use Monte Carlo simulations to calculate the processor reliability. Specifically, we make use of Monte Carlo simulations to combine the effects of the individual failure mechanisms and find the reliability of a single core.

Using Monte Carlo simulations, we first generate a normally-distributed random number, r_{normal}, with mean 0 and standard deviation of 1 using two independent uniformly distributed random numbers r_1 and r_2. Then, we obtain a scaled normally-distributed random number $r_{snormal}$ with mean μ and standard deviation of σ from the normally-distributed random number as follows:

$$r_{normal} = sin(2\pi r_1)\sqrt{-2\ln(r_2)} \tag{3}$$

$$r_{snormal} = \mu + r_{normal}\sigma \tag{4}$$

Then, a random lognormal distribution number $r_{lognormal}$ representing a random lifetime for each failure mechanism can be generated from the scaled normal random number:

$$r_{lognormal} = e^{r_{snormal}} \tag{5}$$

Mean of the normal distribution $r_{snormal}$ (μ) and the mean of the lognormal distribution $r_{lognormal}$ (MTTF) are related to each other as follows:

$$\mu = \ln(MTTF) - \frac{\sigma^2}{2} \tag{6}$$

In order to compute the reliability of a processor which is composed of lognormally distributed failure mechanisms, we generate $r_{lognormal}$ distributions (i.e., random lifetimes) for each failure mechanism. To compute $r_{lognormal}$, we first calculate MTTF values using Eqn. 1 for each failure mechanism and calculate μ using Eqn. 6. We conduct the experiment for 10^6 iterations to generate random lifetimes for failure mechanisms. At each iteration, the lifetime of the processor is set to the minimum of the generated numbers. MTTF of the processor is then calculated by averaging the minimums. To convert the MTTF value to reliability, we generate the cumulative distribution function (CDF) of lognormal distribution. The reliability over time t for the lognormal distribution is then determined by Eqn. 7, where $F(t)$ is the CDF of lognormal distribution at time t.

$$R_t = 1 - F(t) \tag{7}$$

B. System Reliability Modeling

Most prior work on system reliability modeling considers series systems [8, 20], where the first failure on any unit on the chip causes the entire processor to fail. Real-life computer systems, however, may have different levels of series-parallel topologies. In a *series* system of n components, the system fails if *any* of its components fails. On the other hand, a *parallel* system with n components fails if *all* of its components fails. Assuming failure rates are statistically independent, the overall system reliability of a series/parallel topology containing n cores can be computed as follows:

$$Series: \quad R_{system}(t) = \prod_{i=0}^{n} R_i(t) \tag{8}$$

$$Parallel: \quad R_{system}(t) = 1 - \prod_{i=0}^{n} (1 - R_i(t)) \tag{9}$$

IV. Topology and System Reliability Analysis

To explore the effects of system topology on reliability, we consider an 8-core system that has two processors (each with 4 cores) in two separate sockets. Each processor has two chips put together in a single package. Figure 1 provides a diagram of the target system, which is based on Intel Clovertown.

We investigate the following reliability topologies: (a) all 8 cores connected in series; (b) cores in series within each processor, parallel across processors (processor-level parallel); (c) cores in series within each chip, parallel across chips (chip-level parallel); and (d) all cores in parallel.

An all-parallel system incurs higher design cost as additional hardware is needed to detect runtime core failures and initiate the recovery process for continued execution. The OS should also be equipped to safely reconfigure the system on failure. Additional design cost would be reduced with scenario (c), where the parallelism is at the chip-level. Processor-level parallelism, as in (b), can be implemented in today's clusters through using sockets that allow replacement of failed processors or using multiple server nodes.

We next compare *thermal balancing* (e.g., [1, 22]), where high-power loads are distributed across the chip, against *clustering*, where power-hungry loads are allocated on neighboring cores. For each scenario, cores are assigned high (T_H) or low (T_L) temperatures. In clustered mode, cores 0, 1, 2 and 3 have T_H and cores 4, 5, 6 and 7 have T_L. In balanced mode, cores 0, 2, 4 and 6 have T_H and the rest of the cores have T_L. However, in balanced mode, heat transfer between adjacent cores should be taken into account; thus, we assign T_B, the average of T_H and T_L, to all cores. This approximation has a few degrees error compared to detailed temperature simulations, but is sufficient to demonstrate the trends.

Figure 2 compares the system reliability of clustered and balanced modes for each topology. High temperature is set as $80°C$ and low temperature is swept from $40°C$ to $70°C$. Clustering degrades system reliability for *all series* scenario

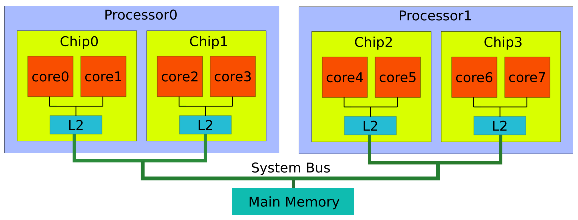


Fig. 1: Layout of the Intel Clovertown System [21].

due to higher core temperatures. However, clustering improves reliability significantly for *processor-level parallel* system and moderately for *chip-level parallel* system. For *processor-level parallel* case, clustering provides system reliability of 0.999 and 0.995 for T_L values of 40°C and 50°C, respectively. For T_L of 60°C, it increases the system reliability from 0.2 to 0.8. Maximum increase in reliability (from 0.073 to 0.429) is seen at T_L of 65°C, which corresponds to 4.85X improvement. As the level of parallelism increases, system reliability for both clustered and balanced modes gets higher. Therefore, for *chip-level parallel* case, clustering is advantageous only at higher T_L values; while for *all parallel* case, it provides almost no improvement. In the rest of the paper, due to its ease of real-life implementation compared to other parallelism scenarios, we focus on *processor-level parallel* systems.

V. Reliability Optimization Policy

Section IV shows that *clustering* provides considerable reliability improvements in *processor-level parallel* and *chip-level parallel* systems compared to thermal balancing. Motivated by this analysis, we propose a topology-aware reliability optimization policy, *Globally Clustering Locally Balancing (*GCLB*)*, where global refers to decisions across parallel nodes, and local refers to allocation decisions among a set of series nodes (e.g., cores within a processor). In this work, we focus on the *processor-level parallel* scenario, as it is commonly employed in real-life multi-chip multicore systems.

We present a flow chart illustrating the *GCLB* optimization policy in Figure 3. The policy periodically polls the performance counters and predicts the power consumption of each application using counter data. Then, we assign the jobs to cores according to their predicted power following the *GCLB* algorithm. The main idea of the algorithm is globally clustering high-power applications among parallel multicore processors and performing thermal balancing locally within a processor. This is because clustering across parallel nodes improves reliability; whereas for a set of series components, balancing results in higher reliability.

We check new job arrivals at every 10ms, which is the the typical scheduler tick in today's OSes. We select a larger interval for GCLB, i.e., 50ms, to limit the performance impact of the policy. At 10ms intervals, we make intermediate heuristic decisions for job allocation. At 50ms intervals, the policy re-arranges the load across the processors if needed by migrating

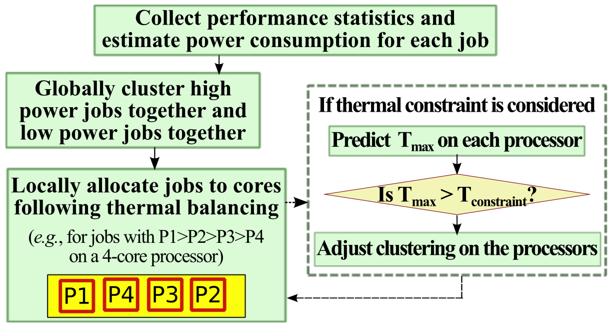
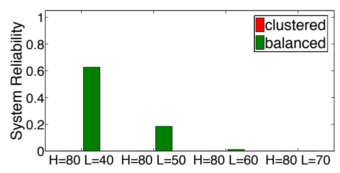
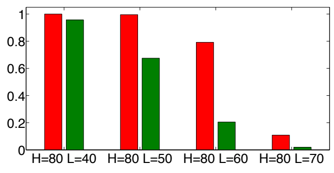

Collect performance statistics and estimate power consumption for each job
Globally cluster high power jobs together and low power jobs together
Locally allocate jobs to cores following thermal balancing
(e.g., for jobs with P1>P2>P3>P4 on a 4-core processor)
P1
P4
P3
P2
If thermal constraint is considered
Predict T_{max} on each processor
Is $T_{max} > T_{constraint}$?
Adjust clustering on the processors

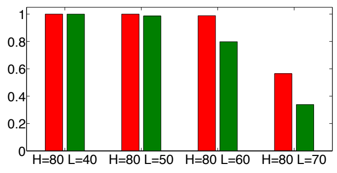
Fig. 3: A flow chart for illustrating the *GCLB* reliability optimization policy for *processor-level parallel* systems.



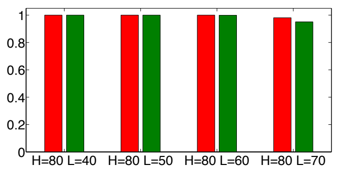
(a) All series



(b) Processor-level parallel



(c) Chip-level parallel



(d) All parallel

Fig. 2: System reliability for different series-parallel scenarios with T_H=80°C and per-core MTTF of 5 years at 60°C.

applications. Prior work has reported that cold-start overhead dominates the migration cost for SPEC benchmarks, and the total migration overhead is less than 1 ms [1]. Assuming a similar overhead in our system, a scheduling interval of 50ms causes maximum 2% performance cost.

At every 10ms, we assign newly arriving jobs to the idle cores on the system. To cluster higher power loads, we first assign new jobs to processors with a higher average power. If there is a thermal constraint, we predict the maximum processor temperature for the processor running the new job. If the maximum temperature is exceeded, we assign the new job to the processor with the next highest average power.

At every 50ms, we apply the GCLB policy. Assuming the system has m cores, l parallel processors, and there are n jobs to be allocated (we assume $n \leq m$), we first estimate the power consumption for each job on the system. Then, we sort the power values for all the jobs. We group the sorted jobs into l groups: jobs with the highest power values are assigned to the first processor, the group with the second largest power values in the queue are assigned to the second processor, etc., until all the jobs are allocated.

Once the jobs are clustered across parallel processors, within each processor, we locally balance the temperature across the cores (i.e., across series components). The balancing method is based on thermal balancing policies in prior work [1], where high power jobs are assigned to expected cool locations on the chip, such as corner or side cores. Cooler jobs run in the central area, which is generally hotter. Figure 4 demonstrates the global clustering and balancing policies. Thermal balancing is applied to each processor locally.

Power Prediction:

To estimate power consumption of each job, we collect performance statistics. We track instructions per cycle (IPC), number of floating-point instructions, and number of integer instructions, as these metrics are strongly correlated with power consumption [23]. We collect the performance data using a simulator in our evaluation, while in a real system the statistics are collected through performance counters. We build a linear equation of the three performance counters using regression, and predict power consumption based on the equation. Experiments with 17 SPEC benchmarks show

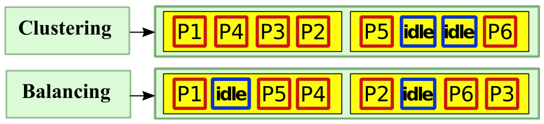


Fig. 4: An illustration of the clustering and balancing job allocations on the target system under 75% utilization. P represents power consumption, and $P1 > P2 > ... > P6$.

4% prediction error using this method. Performance impact of power prediction is negligible, since computing a simple equation has very low computational cost.

Working with Thermal Constraints:

Several recent techniques focus on accurate runtime temperature prediction (e.g., [24]). In this work, we choose a simple temperature prediction method using a linear model as we solely want to estimate the maximum temperature on a processor. For inputs to the predictor, we use power estimates for each core and absolute power differences between adjacent cores to take the heat sharing and core locations into account. We collect 100 sets of simulation results from the SPEC 2006 workloads, and validate the predictor against HotSpot simulations. Our peak temperature prediction results in maximum 8% error in comparison to HotSpot simulation results, with less than 2^oC error for most cases.

GCLB algorithm can work with temperature constraints using the thermal predictor. This is important as clustering high-power workloads may result in high peak temperatures on a processor. In addition to critical thermal thresholds determined by the manufacturer, thermal constraints could be imposed by user-defined target per-core MTTF values or by cooling optimization policies. During allocation, if the thermal constraints are not satisfied, we adjust job allocation by swapping the hottest jobs across processors and locally balance temperature after swapping. This process is repeated (a job moved once is not moved again) until the thermal constraint is met. In this paper, we assume we can always find a schedule that meets thermal constraints, which is a reasonable assumption for most commercial systems.

The proposed GCLB policy can also be integrated with DVFS policies. Integration with DVFS can provide energy savings as well as fine tuning of the operating conditions to meet temperature or performance constraints. Hybrid policies integrating various DVFS and job allocation strategies have been designed in prior work [1].

VI. Experimental Results

We model the target system based on the core microarchitecture of Intel Clovertown. The architectural parameters for cores and caches are listed in Table I. We use M5 [25] to build the performance simulation infrastructure. We use the system-call emulation mode in M5 with X86 instruction set architecture (ISA). We fast-forward each benchmark for 1 billion instructions for warm up and execute with the detailed out-of-order CPUs for 100 million instructions.

We select 17 applications from the SPEC 2006 benchmark suite, among which 10 applications are integer (INT) benchmarks (*astar, bzip2, gcc, gobmk, h264ref, hmmer, libquan-*

TABLE I: Core Architecture Parameters.

CPU Clock	2.66 GHz
Issue Width	4-way out-of-order
Functional Units	3/2 Int/FP ALU, 1/1 Int/FP Mult
Physical Regs	128 Int, 128 FP
RAS / ROB size	16 /96 entries
Load /Store Queue	32 / 20 entries
L1 I/DCache	32 KB, 8-way, 64B-block
L2 Cache(s)	4 MB, 16-way, 64B-block

tum, mcf, omnetpp, specrand_int) and 7 applications are floating point (FP) benchmarks (*bwaves, cactusADM, dealII, GemsFDTD, lbm, namd, specrand_fp*). We further classify these benchmarks according to their performance and memory boundedness. They are named *INT-Hmem, INT-Lmem, INT-HIPC, INT-LIPC, FP-Hmem, FP-Lmem, FP-HIPC, FP-LIPC, and Mixed*, where *Hmem* or *Lmem* means workloads with high or low memory access rates, *HIPC* or *LIPC* means workloads with high or low IPC.

We use McPAT 0.7 [26] for 65nm process to obtain the runtime dynamic power of the cores. We set V_{dd} to 1.1V and operating frequency to 2.66GHz. The L2 cache (4 MB) power is calculated using Cacti 5.3 [27] as 5.06W. We calibrate the McPAT run-time dynamic core power using the published power for Intel Xeon Processor X5355. At 343K, we assume the leakage power for the cores is 35% of the total core power. We also model the temperature impact on leakage power using an exponential formula [8].

We run HotSpot 5.0 [28] for thermal simulations. We set the chip and package parameters using the default configuration in HotSpot to represent efficient packages in high-end systems. All simulations use the HotSpot grid model for higher accuracy and are initialized with the steady-state temperatures. The chip and core areas are obtained from the published data for Intel Clovertown systems. The L2 cache area is estimated by using Cacti 5.3 [27].

We next evaluate *GCLB* on the target Intel Clovertown system for different utilization scenarios. High performance computing clusters are examples of computer systems with high utilization. Figure 4 compares the clustering and balancing allocation policies at 75% utilization. System reliability of the clustering and balancing policies for all the workloads running on the target system with 75% workload utilization is shown in Figure 5. We observe that the proposed *GCLB* policy provides up to 123.3% improvement in system reliability compared to the thermal balancing policy. Among all the workloads, the $HIPC$ and $Lmem$ applications have higher system reliability improvement. This is because the $HIPC$ and $Lmem$ applications have higher power densities causing higher temperatures. Local thermal balancing has up to 27.2% reliability improvement compared to not balancing allocation within a processor. As local balancing always outperforms locally imbalanced scenarios, we do not report results for locally imbalanced cases in the rest of the results.

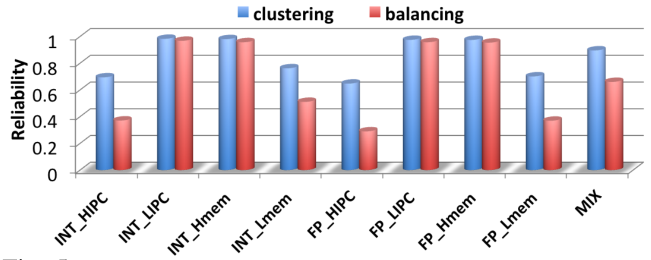


Fig. 5: System reliability with *GCLB* (clustering) and thermal balancing allocation policies for the target system under 75% utilization.

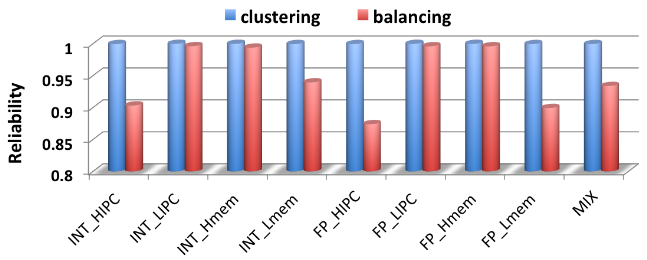


Fig. 6: System reliability with *GCLB* and thermal balancing allocation policies for the target system under 50% utilization.

Figure 6 shows the system reliability for the clustering and balancing allocation policies on the target system with 50% workload utilization (utilization level similar to data centers). The job allocations for the 50% workload utilization is similar to the illustration shown in Figure 4, while the $P5$ and $P6$ change to idle cores. We see that with 50% workload utilization, we achieve up to 14.3% improvement in the system reliability in comparison to thermal balancing policy. We also conduct the same analysis on the target system with 25% workload utilization. The low workload utilization scenario happens when data centers run fewer jobs (e.g., at night). In this case, clustering and balancing achieve similar reliability.

When *GCLB* is applied without considering thermal constraints, peak temperature at 75% utilization is between 63.8^oC and 76.33^oC. Figure 7 illustrates the system reliability with *GCLB* optimization policy compared to the thermal balancing policy at 75% utilization, using a thermal constraint of 75^oC. We notice that the reliability improvement of *GCLB* decreases for some workloads, such as FP_HIPC. This is because GCLB moves some of the higher power jobs to lower power processors to meet the constraint, and becomes more similar to balancing.

We also explore the *GCLB* policy for dynamically changing workloads. We generate a random workload utilization scheme which changes every 10ms with a total simulation time of one second. The average workload utilization is 68%. The jobs running on the system are randomly selected among the 17 SPEC benchmarks. Figure 8 shows that allocating jobs according to *GCLB* policy improves reliability by 27.3% on average compared to random workload allocation. Figure 8 also shows that, if the *GCLB* optimization policy is applied every 10ms without considering thread migration overhead, the

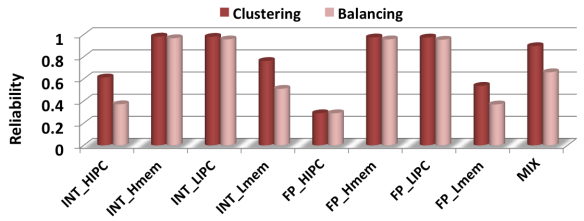


Fig. 7: System reliability for *GCLB* optimization policy compared to thermal balancing for systems with 75% utilization, considering a thermal constraint of 75^oC.

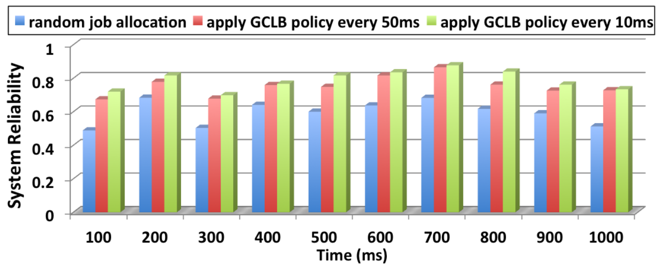


Fig. 8: System reliability of *GCLB* compared to random job allocation for dynamically changing workload utilization.

average system reliability improvement is 32.9%. However, as discussed in Section V, migrating threads every 10ms would cost up to 10% system performance overhead. Our reliability optimization policy achieves comparable reliability improvement with less than 2% performance cost.

We extend our analysis to a 16-core system with 4 parallel processors and 4 cores (in series) on each processor. System reliability for the 16-core system running *GCLB* compared to thermal balancing is presented in Figure 9. We observe that *GCLB* policy provides system reliability of close to 1 for all the benchmarks, and improves reliability by up to 101.7% in comparison to thermal balancing. This is because scaling to a higher number of processors provides increased parallelism and higher degree of freedom for more efficient task scheduling. For example, for the 16-core system with 75% utilization, using "clustering" assigns all the "idle" cores in one processor, which increases system reliability.

VII. Conclusion

In this paper, we have proposed a topology-aware job scheduling policy that optimizes the reliability of multi-chip multicore systems. We have shown that thermal balancing policies improve the reliability for systems with series components. When reliability topology is considered in systems with parallel components, clustering high power jobs outperforms balancing. Our `GCLB` policy clusters jobs globally in a system across parallel nodes and balances jobs locally within a processor. It is also able to consider thermal constraints and adapts to workload changes.

We have evaluated our policy under various workload scenarios. HPC systems and data centers are typically under medium to high utilization. Under such conditions, our policy improves the reliability of multi-chip systems by up to 123.3% compared to thermal balancing. We have also shown that under dynamically changing workloads, `GCLB` improves system reliability by 27.3%. Finally, we have studied our policy's

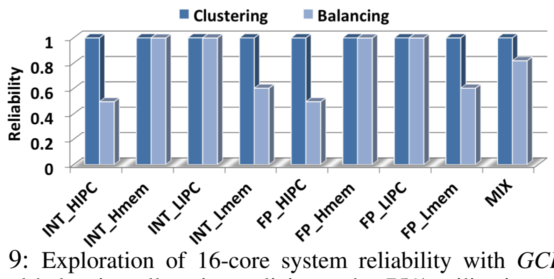


Fig. 9: Exploration of 16-core system reliability with *GCLB* and thermal balancing allocation policies under 75% utilization.

scalability. When the size of the target system is doubled, `GCLB` improves system reliability by up to 101.7%.

Acknowledgements

This work has been partially funded by NSF grant CNS-1149703 and Sandia National Laboratories.

References

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

[2] J. Donald and M. Martonosi, "Techniques for multicore thermal management: Classification and new exploration," in *ACM SIGARCH Computer Architecture News*, vol. 34, no. 2, 2006.

[3] Z. Lu, J. Lach, M. Stan, and K. Skadron, "Improved thermal management with reliability banking," *IEEE Micro*, vol. 25, no. 6, 2005.

[4] R. Teodorescu and J. Torrellas, "Variation-aware application scheduling and power management for chip multiprocessors," *ACM SIGARCH Computer Architecture News*, vol. 36, no. 3, 2008.

[5] J. Winter and D. Albonesi, "Scheduling algorithms for unpredictably heterogeneous cmp architectures," in *DSN*, 2008, pp. 42–51.

[6] H. Hanson *et al.*, "Thermal response to dvfs: Analysis with an intel pentium m," in *ISLPED*, 2007, pp. 219–224.

[7] A. Kumar *et al.*, "Hybdtm: a coordinated hardware-software approach for dynamic thermal management," in *DAC*, 2006, pp. 548–553.

[8] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "The case for lifetime reliability-aware microprocessors," in *International symposium on Computer Architecture (ISCA)*, 2004, pp. 276–287.

[9] S. Biswas *et al.*, "Fighting fire with fire: modeling the datacenter-scale effects of targeted superlattice thermal management," in *ISCA*, 2011.

[10] V. Hanumaiah and S. Vrudhula, "Reliability-aware thermal management for hard real-time applications on multi-core processors," in *Design, Automation Test in Europe Conference (DATE)*, March 2011, pp. 1–6.

[11] S. Wang and J.-J. Chen, "Thermal-aware lifetime reliability in multicore systems," in *ISQED*, 2010, pp. 399–405.

[12] L. Huang, F. Yuan, and Q. Xu, "Lifetime reliability-aware task allocation and scheduling for mpsoc platforms," in *DATE*, 2009, pp. 51–56.

[13] J. Srinivasan *et al.*, "Exploiting structural duplication for lifetime reliability enhancement," in *ISCA*, 2005, pp. 520–531.

[14] J. Moore *et al.*, "Making scheduling "cool": temperature-aware workload placement in data centers," in *USENIX*, 2005, pp. 5–15.

[15] A. K. Coskun, T. Rosing, K. Whisnant, and K. Gross, "Static and dynamic temperature-aware scheduling for multiprocessor socs," *IEEE Trans. VLSI Systems*, vol. 16, no. 9, pp. 1127–1140, 2008.

[16] L. Wang *et al.*, "Towards thermal aware workload scheduling in a data center," in *I-SPAN*, 2009, pp. 116–122.

[17] "Failure mechanisms and models for semiconductor devices, jedec publication jep122e." [Online]. Available: http://www.jedec.org/

[18] M. Alam *et al.*, "A comprehensive model for pmos nbti degradation: Recent progress," *Microelectronics Reliability*, vol. 47, no. 6, 2007.

[19] K. Ferreira *et al.*, "Evaluating the viability of process replication reliability for exascale systems," in *SC*, 2011, pp. 1–12.

[20] Y. Xiang *et al.*, "System-level reliability modeling for mpsocs," in *CODES/ISSS*, 2010, pp. 297–306.

[21] Q. Teng, P. F. Sweeney, and E. Duesterwald, "Understanding the cost of thread migration for multi-threaded java applications running on a multicore platform," in *ISPASS*, 2009, pp. 123–132.

[22] F. Mulas *et al.*, "Thermal balancing policy for streaming computing on multiprocessor architectures," in *DATE*, 2008, pp. 734–739.

[23] T. Li and L. K. John, "Run-time modeling and estimation of operating system power consumption," in *SIGMETRICS*, 2003, pp. 160–171.

[24] R. Z. Ayoub and T. S. Rosing, "Predict and act: dynamic thermal management for multi-core processors," in *ISLPED*, 2009, pp. 99–104.

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

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

[27] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi, "CACTI 5.1," HP Laboratories, Palo Alto, Tech. Rep., April 2008.

[28] K. Skadron, M. R. Stan, W. Huang, V. Sivakumar, S. Karthik, and D. Tarjan, "Temperature-aware microarchitecture," in *ISCA*, 2003.