NBTI Aware Workload Balancing in Multi-core Systems

Jin Sun*, Avinash Kodi†, Ahmed Louri*, and Janet M. Wang*
*Department of Electrical and Computer Engineering, University of Arizona
1230 E. Speedway, Tucson, Arizona 85721, Email: {sunj,louri,wml}@ece.arizona.edu
†Department of Electrical and Computer Engineering, Russ College of Engineering and Technology, Ohio University
322D Stocker Center, Athens, Ohio 45701, Email: kodi@ohio.edu

***Abstract*—As device feature size continues to shrink, reliability becomes a severe issue due to process variation, particle-induced transient errors, and transistor wear-out/stress such as Negative Bias Temperature Instability (NBTI). Unless this problem is addressed, chip multi-processor (CMP) systems face low yields and short mean-time-to-failure (MTTF). This paper proposes a new design framework for multi-core system that includes device wear-out impact. Based on device fractional NBTI model, we propose a new NBTI aware system workload model, and develop new dynamic tile partition (DTP) algorithm to balance workload among active cores while relaxing stressed ones. Experimental results on 64 cores show that by allowing a small number of cores (around 10%)to relax in a short time period (10 second), the proposed methodology improves CMP system yield. We use the percentage of core failure to represent the yield improvement. The new strategy improves the core failure number by 20 %, and extend MTTF by 30% with little degradation in performance (less than 6%).**

I. Introduction

As device feature size continues to shrink, reliability becomes a severe issue due to process variation, particle-induced transient errors, and transistor wear-out/stress such as the Negative Bias Temperature Instability (NBTI) that affects system life-span. Typical strategies applied to the manufacture process include regulating the voltage supply, controlling temperature and monitoring frequency. The above measures have provided robust designs for several technology generations ranging from 0.5 um to 0.10 um. However, as feature size continues to shrink reaching the nanometer scale, these techniques will no longer be sufficient in the nanometer regime [1][2][3]. The extremely small device feature size makes process variation, particle-induced transient errors, and device stress much more challenging. Unless these issues are effectively addressed, chip multi-processor (CMP) systems will face low yields and short mean-time-to-failure (MTTF)[1].

A number of techniques have emerged recently to deal with process variation and transient errors. These include detecting and correcting transient defects occurred in memory storage, using time-redudant computation (TRC), error correction codes (ECC), and dual or triple-modular redundancy (DMR or TMR). Additional work focused on including design tolerance for permanent faults such as NBTI using post-manufacturing burn-in [1]. However, very little attention has been paid to device stress and its impact on system life-span and performance in the multi-core era. This paper proposes a new design framework for multi-core system to include device stress impact. It first introduces device fractional NBTI model, and proposes a new NBTI aware system workload model. This is followed by a new dynamic tile partition (DTP) algorithm to balance workload for active cores while relaxing stressed ones.

Device stress may happen after days of full workload operation, and requires days to relax before recovering. Letting the device completely wear-out will impact the system as defective cores have to be removed from the pool of active cores. Different from existing approaches [2][3] that focused on long term stress using static NBTI models, we propose to use fractional stress and recovery in the multi-core systems. We model NBTI with regard to threshold voltage changes, and then reflect the threshold variations to timing degradation of the core. By deploying a fractional NBTI model where cores are partially stressed before complete wear-out, the overall number of active cores will be much higher than in the complete stressed out model. Base on the fractional NBTI model, we propose a strategy where the cores have to alternate between a full workload phase (when the core is relaxed) and a light workload phase (when the core is stressed) with high frequency of alternation between phases. Fig. 1 displays an example of 4×4 cores with difference workload numbers based on NBTI model at one time point. The strategy

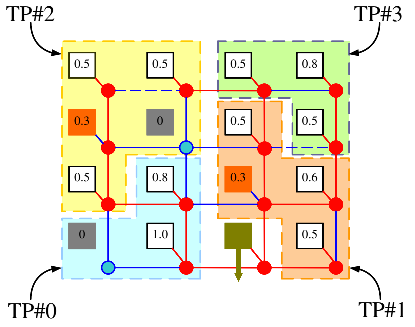


Fig. 1. An example of workload distribution on a 4×4 multi-core system

has shown good system yield improvement and extension of MTTF with insignificant penalty on system throughput, and no impact on latency and traffic overhead. Specifically, this new contributions of this paper are: 1) a system NBTI stress model, 2) a new Dynamic Tile Partitioning (DTP) algorithm, 3) a first insight into the relationship among core recovering time, stress time and workload, and their impact on core life-span. Experimental results on 64 cores show that by allowing a small number of cores (10%)to relax in a short time period (10 second), the proposed methodology improves CMP system yield by reducing core failure percentage about 20 %, and extend s MTTF by 30% with little degradation in performance (less than 6%).

The remainder of this paper is organized as follows. Section II introduces NBTI device and core model. Section III describes workload balancing with DTP approach. Section IV discusses experimental results. Finally, section V concludes the paper.

II. NBTI MODEL FOR MULTI-CORE PERFORMANCE

As one of the main reliability issue, NBTI limits lifetime in nano-scale integrated circuits. When PMOS is negatively biased, the electrical field across gate oxide produces a complicated electro-chemical reactions that consequently increases PMOS threshold voltage over time. Manifesting itself in a gradual manner, the impact of NBTI may take days and months to affect timing and circuit delay and thus fail the system. Recent research works have confirmed that NBTI is getting worse with further scaling starting from 90nm technology[1][2][3].

According to [2], the widely different diffusivity of H_2 in the oxide and polysilicon causes PMOS transistors to alternate between stress and recovery. The recovery itself consists of two steps, a fast recovery and a slow recovery. Fig. 2 demonstrates the alternation of stress and recovery. In general, recovery and stress period are fairly symmetric. We use 10 second for each period. During the recovery stage, the first 1 second, the device already recovers 90% of the stress. the later 9 second recovers the rest 10%. The value 10 second per period comes from the sampling time of NBTI sensors [5]. It has been reported as accurate enough to monitor the NBTI introduced threshold voltage changes. The NBTI compact device model follows the similar style as reported in [3]. The stress model is derived as:

$$\Delta V_{th} = (K_v(T(t))(t-t_0)^{\frac{1}{2}} + \Delta V_{th0}{}^{\frac{1}{2n}})^{2n} \quad (1)$$

The recovery model can be written as:

$$\Delta V_{th} = V_{th0}(1 - \frac{2\epsilon + \sqrt{\epsilon_2 C(T(t))(t-t_0)}}{2t_{ox} + \sqrt{C(T(t))t}}) \quad (2)$$

However, we also consider the temperature changes with regard to time. That is, we include the temperature dependency on time in both coefficient K_v and C. This inclusion is to avoid inaccuracy when temperature changes with regard to time. The changes in threshold voltage in turn affect timing and power of the core. According to [4][6], gate delay model considering threshold voltage may be described as a first-order Taylor expansion. We extend this model to critical path delay model. That is, the i-th critical path delay can be written as

$$d_i = d_i(V_{th0}, L_{eff}) + (\partial d_i/\partial V_{th})\Delta V_{th} \quad (3)$$

Here the delay is modeled as Gaussian distribution with $d_i(V_{th0}, L_{eff})$ as the nominal delay value. For one single core, we may have a number of critical paths. The worst case is the critical path with the biggest variation. Hence,

$$\Delta d_{max} = min(max((\partial d_i/\partial V_{th})\Delta V_{th}), 3\sigma) \quad (4)$$

The bigger the delay variations, the less workload the core should accept. We propose using the following percentage model to relate delay variations with workload. Assume when delay is at nominal value and has zero variation, the workload is at 100% or 1. And at 3σ of the delay variation distribution, the workload is at lowest, 0. Then the percentage of workload is modeled as

$$workload_{delay} = 1 - \Delta d_{max}/3\sigma \quad (5)$$

It is apparent that our concern is positive delay changes, or delay is bigger than nominal value. Here, when Δd_{max} is zero, we have full workload. The opposite situation is that Δd_{max} is equal to 3σ. In this case, we have zero workload. The majority workloads will lie in between these two extreme cases.

While V_{th} has little impact on dynamic power consumption, it affects dramatically the leakage power. Using empirical leakage power model, we have

$$P_{leak} = C_0 exp(-C_1 L_{eff} - C_2 V_{th}) \quad (6)$$

Substituting the first order pertubation model for $V_{th} = V_{th0} + \Delta V_{th}$, this equation can be further written as

$$P_{leak} = C_0 exp(-C_1 L_{eff} - C_2 V_{th0}) exp(-C_2 \Delta V_{th}) \quad (7)$$

The normalized leakage power is defined as

$$P_{nleak_i} = \frac{P_{leak}}{C_0 exp(-C_1 L_{eff} - C_2 V_{th0})} \quad (8)$$

While these equations work for single transistor, for every gate and every core, the total leakage power is simply the sum of leakage power of all transistors in one gate or one core. That is,

$$P_{leakTotal} = \sum_{k=1}^{N} P_{nleak_i} \quad (9)$$

Again, we identify leakage extreme case comparing with its $3\sigma_{leak}$ value.

$$\Delta P_{max} = min(P_{leakTotal}, 3\sigma_{leak}) \quad (10)$$

The impact on workload may be formulated as

$$workload_{leakP} = 1 - \Delta P_{max}/3\sigma_{leak} \quad (11)$$

If we consider both delay and leakage power at the same time, a simple model provides

$$workload = 0.5 * workload_{delay} + 0.5 * workload_{leakP} \quad (12)$$

This model averages leakage and delay impact on workload. Therefore, we have transitioned from single PMOS NBTI model to workload intepretation. During this process, the NBTI device model first provided its impact on threshold voltage. Then we model the variation of threshold voltage on system level delay and leakage power respectively. The end result is the workload for each core in percentage (less than 1). The workload value generated will be treated as upper bound limit for each core. Note that NBTI impact on threshold voltage changes with regard to time, so as the workload. That is, at different time points, the workload upper bound for each core would be different as well.

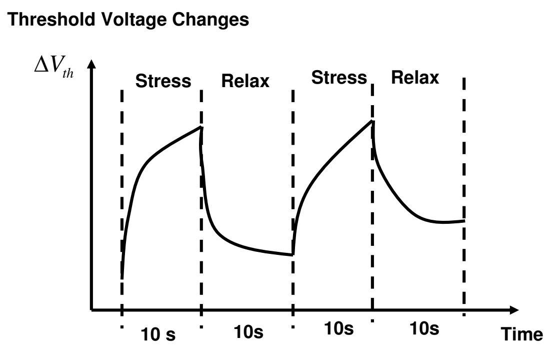


Fig. 2. NBTI PMOS Fractional Model with Both Stress and Relaxing Phases

III. NBTI Aware Workload Balancing

This section investigates the design methodology of workload and traffic balancing techniques considering NBTI. It consists of three components: Dynamic Tile Partitioning (DTP), task scheduling within each partition, DTP based workload balancing and traffic load balancing. Based on workload number suggested by NBTI system model, the new method assigns cores into different partitions with comparative total workload. Then in the presence of active flows, the new method determines the assignment scheduling for each core, in order to reflect the tasks of the flows into particular cores within the assigned partition. By introducing penalty table for each router, the proposed method distributes traffic in the network uniformly across the entire network to avoid overload of some portion of the network.

A. Dynamic Tile Partitioning

The concept of Dynamic Tile Partitioning (DTP) is the key to our proposed workload balancing framework. The purpose of DTP is to spread out the tasks of different flows across the entire network. This helps to avoid the overstress on a small portion of the network, and to further improve the efficiency and reliability of the entire multi-core system [7][8]. The basic idea of tile partitioning is to organize all the cores into several necessary partitions. The processing of tasks of a particular flow is restricted to an assigned partition, consisting of a set of tiles physically adjacent to each other. Based on the assigned workload numbers using NBTI system model, DTP divides the cores into different tile partitions with comparative performance measurements. Fig. 1 explains the concept of tile partitioning with an example. The diagram shows a 4×4 tile-based multi-core network which was partitioned into 4 tile partitions. Each core in the grid is labeled with a workload number. The cores in gray color represent the ones have NBTI wear-out, whose workload numbers are denoted by low values. DTP groups all the cores into several tile partitions (distinguished by respective colors), in order to make sure the sum of workload numbers within each partition has nearly the same amount. In other words, each partition has (nearly) equal-quantized workload. The example in Fig. 1 indicates that tile partitions TP#0, TP#2 and TP#3 have identical total workload numbers of 1.8, while the workload number of tile partition TP#1 is 1.9, which is very close to those of other partitions. Given the workload number W_i for the i-th core. Our objective is to develop a partitioning scheme that, within each partition the sum of included workload numbers has to be nearly the same amount (of course all the elements in each partition must be mutually connected). More precisely, if there are K partitions $C_1, C_2, \cdots, C_K$, the sum of all the workload numbers for each partition is accordingly $W_{sum}^{(1)}, W_{sum}^{(2)}, \cdots, W_{sum}^{(K)}$, DTP attempts to guarantee the values of all the $W_{sum}^{(j)}$'s have very small differences. Or in other words, we need to minimize the variance of sequence $\{W_{sum}^{(1)}, W_{sum}^{(2)}, \cdots, W_{sum}^{(K)}\}$.

The DTP algorithm could be implemented in the following searching steps:

Step 1. Set a threshold, or tolerance value for the variance of sequence $\{W_{sum}^{(1)}, W_{sum}^{(2)}, \cdots, W_{sum}^{(K)}\}$. We call this tolerance value σ_t, which is used for comparison during each searching step.

Step 2. Start with an arbitrarily selected partitioning solution (with connectivity guaranteed). Intuitively, we can start with the uniform or symmetric grouping scheme.

Step 3. Compute the variance of this initialized grouping. If it is smaller than the predetermined tolerance value σ_t, the searching procedure is stopped. Otherwise, go to step 4.

Step 4. Give small adjustment to above grouping result, and repeat N times randomly. We will obtain a set of newly generated partitioning solutions. Compute the variance of sequence $\{W_{sum}^{(1)}, W_{sum}^{(2)}, \cdots, W_{sum}^{(K)}\}$ for each newly generated partitioning solution, and compare it with the tolerance value σ_t. If the variance is smaller than σ_t, the searching procedure is stopped. Otherwise, go to step 5.

Step 5. For all these N grouping results generated in step 4, select the optimal one among these solutions (i.e. the grouping solution having the smallest variance) and repeat step 4.

This is a heuristic searching procedure and thus adaptive to the transient update in workload number as well as the number of active flows. In most situations, some of cores are possibly under "quasi-defect" situations, such as overstressed or suffering from transient errors. Under this situation, the overstressed cores can not be assigned tasks at that moment. On the other side, at a later time when they are released, they may be available again. In this sense, the workload number for a core in not a constant number but has to be updated frequently. Another factor which influences the result of DTP is the number of partitions to be organized. The number of partitions is also a time-dependent parameter, which is determined by the number of active flows and the number of available cores. At a particular point, given a multi-core network with fixed workload numbers assigned and determined partition numbers, DTP returns the best fitting solution for tile grouping. The final solution of DTP is not the global optimal or strict optimal solution, but a secondary optimal or local optimal solution (due to the tolerance value we predetermined). However, strict optimality is not necessary in our case, since the eventual objective is to make all the partitions distributed evenly. The general flows of this DTP algorithm can be described by following pseudocode (Fig. 3), where the input parameter network represents a given tile-based multi-core network with measured workload numbers, parameter NP is the number of partitions to be organized, and σ_t denotes the predetermined variance threshold.

```
Algorithm DTP (network, NP, $\sigma_t$)

initialize Initial_grouping ← Symmetric_grouping (network, NP)
Temp_$\sigma$ ← Compute_Variance (Initial_Grouping)
if Temp_$\sigma \leqslant \sigma_t$
    return (Initial_Grouping)
end if
while (true)
    SC ← {Ø}
    for i = 1:N
        Temp_Grouping ← Perturbation (Initial_Grouping, i)
        Boolean Connectivity ← Check_Connectivity (Temp_Grouping)
        if Connectivity == NO
            continue
        else
            Temp_$\sigma$ ← Compute_Variance (Temp_Grouping)
            if Temp_$\sigma \leqslant \sigma_t$
                return (Temp_Grouping)
            end if
            PUSH (SC, Temp_Grouping)
        end if
    end for
        Index ← Find_Min_Variance (SC)
        Intial_Grouping ← SC (Index)
end while
```

Fig. 3. The pseudocode for the Dynamic Tile Partitioning algorithm

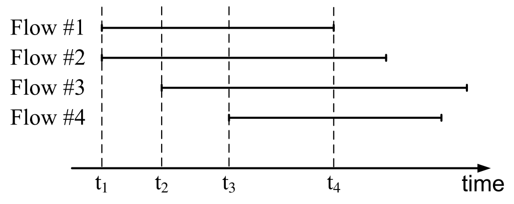
After organizing the tiles into several necessary partitions, the DTP algorithm has to further address the problem of scheduling the cores within the desired partition to process the assigned flow tasks. In other words, we need to determine which cores within this selected partition the task of a particular flow will be assigned to. This is a well-studied task scheduling problem and is discussed elaborately in a number of literatures [9]-[13].

In general, a multi-core system is able to handle a limited number of flows in parallel. Tasks within a particular flow, however, may have dependencies among each other. The precedence relationships among tasks prohibit the execution of parallelism. Tasks from different flows, on the other hand, tend to have no dependency, or only few dependencies among them. Thus a particular flow is typically mapped into a chosen tile partition and all the tasks of this flow are scheduled to be processed by cores within the partition. Therefore the number of tile partitions depends on the number of currently active flows. Besides, the partition number is also restricted by the number of available cores in the network. If the number of currently active flows is greater than the number of available cores, flows that have no partition assigned will be buffered in the waiting list until one of the partitions becomes available. Another buffering situation is that there are no idle cores within the partition assigned for a particular flow.

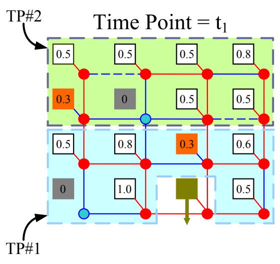
The update of partition number takes places mainly at two moments. The first moment is when a new flow comes in. Here, a flow becomes available, is detected by the system and is taken into account for scheduling immediately to improve the utilization of the schedule. If there are available cores in the system, the number of required partitions will be increased and the cores in the network will also be re-organized according to the updated partition number. Otherwise the new flow should be buffered in the waiting list until the relaxation of certain partition. The second important moment is the time when certain partition finishes processing all the task of the assigned flow. Since the total expected running times of the flows assigned to each partition may differ much, a schedule of one partition may run out of things to do before other partitions complete their assigned tasks. The relaxed cores within this partition must be considered for new assignments in order to prevent waste of time for waiting. The partition may be re-scheduled for the newly coming flow (if there are flows existing in the waiting list); or the cores in the partition shall be re-arranged to join the cores in other partitions for task processing (if the waiting list in the system is empty). Both of these two policies of partition number updating are for the purpose of enhancing the utilization of the processing system and balancing the networking workload.

As long as the number of clusters assigned for the flows has been changed, the DTP algorithm will be performed again and the cores in the network will be re-partitioned for processing. Furthermore, the cores within each partition will also be re-arranged by the scheduling algorithm for task assignments. The previous schedule will be replaced by the newly generated schedule. The previous scheduling of the unexecuted tasks assigned in this partition will be eliminated. However, for the tasks that are currently under processing, we could not disturb the corresponding processing elements. Instead they will be left to finish processing the assigned tasks and re-arranged after the completion of execution. When re-scheduling the remaining tasks of the flows, we need to delete the task nodes that have been finished already and nodes that are in the process of execution from the original DAG. Then based on the reduced DAG and newly grouped cores, we update the Gantt chart and reflect the schedule into the desired partition, by the application of the list scheduling algorithm introduced before.

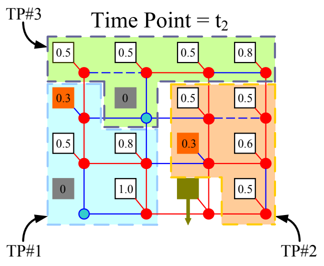
We provide an example as shown in Fig. 4. Here, 4 flows are assigned to a 4×4 tile-based multi-core network. All the cores in the network was labeled with respective workload numbers. Each line segment in the time axis represents the starting time, ending time and duration time for respective flow, as shown in Fig. 4(a). We chose four important time points to explain the proposed workload balancing policy. At time point t_1,



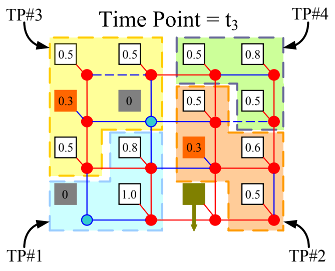
(a) Four active flows in the system



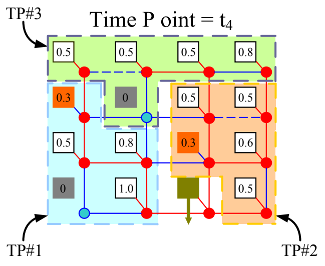
(b) 2 tile partitions at t_1



(c) 3 tile partitions at t_1



(d) 4 tile partitions at t_1



(e) 3 tile partitions at t_1

Fig. 4. Workload Balancing on a 4×4 tile-based multi-core network.

the system detects that two flows #1 and #2 become active and the required number of partitions is initialized as two. The DTP algorithm is performed to organize the cores in the network into two partitions for processing, which is illustrated in Figure 4(b). By the application of list task scheduling within each partition, the tasks of flow #1 and flow #2 are assigned to the cores coming form tile partition (TP) #1 and VC #2 respectively. At time point t_2, the newly detected flow #2 increases the required partition number to three. The DTP algorithm is therefore performed again. The 4×4 network is re-organized into three tile partitions accordingly. Task scheduling on the cores is also updated to substitute the previous scheduling results. When performing re-scheduling for each partition, we leave the cores in execution of tasks to first finish the assigned tasks without disturbing them. They will be re-arranged for processing tasks as long as they become available. In the similar way at time point t_3, flow #4 becomes active. The required partition number is updated to four. The network switches to a 4-partitions grouping and all the cores are re-scheduled for task assignments. Notice that at time point t_4, the last task of flow #1 has been finished. This situation changes the partition number back to three. All the cores within the partition desired to process tasks of flow #1 are relaxed. By updating the core grouping these cores are re-arranged to join other cores to share the networking workload.

IV. Experimental Results

We use 64 cores as a multi-core system example. Table 1 records the result from DTP algorithm considering NBTI aware workload at 90 second after simulation starts. It includes partition number, core index, each core's workload number and its utilization. Only the stressed cores are listed in both tables.

TABLE I
The workload number and core utilization for the stressed cores

Stressed Core Index	Partition Grouped in	Workload Number	Core Utilization
9	1	0.7	0.6313
12	1	0.5	0.4624
28	3	0.5	0.4044
33	3	0.4	0.3679
37	6	0.4	0.3208
39	6	0.5	0.4374
50	7	0.8	0.6722
57	7	0.9	0.8276

To compare system performance, we choose 8/64 nodes to be stressed (12.5%). First, we run simulations at normal load up to 5000 cycles (about 10 second), then reduce the load by 50% every 5000 cycles i.e. at 5000 cycles, the load is now 0.5 of the offered load and at 10000 cycles it will be 25% of the offered load. We terminate at 15000 cycles. "Non-stressed" represents simulation without considering the stress cores and "Stressed" is the opposite one. Fig. 5 shows that the

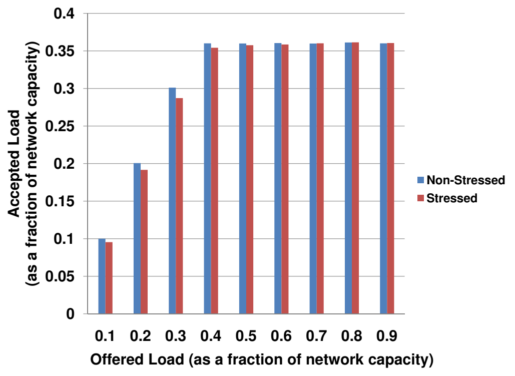


Fig. 5. Comparison of Throughput Result with Non-Stressed and Stressed Cases

throughput almost identical before considering NBTI and after considering NBTI. Here, while stressed load on 8/64 nodes is reduced, other nodes are at full rate. Here an insignificant 6% drop in throughput is observed.

Similar observation can be made with latency in Fig. 6. When workload increases beyond 0.3, very minor differences can be observed between the ideal "Non-stressed" and the "Stressed" cases. As the workload increases, we see that this difference satuarates. This is mainly due to the fact that most packets that could be injected are injected early in the network simulation. Therefore, these packets form the bulk of the traffic.

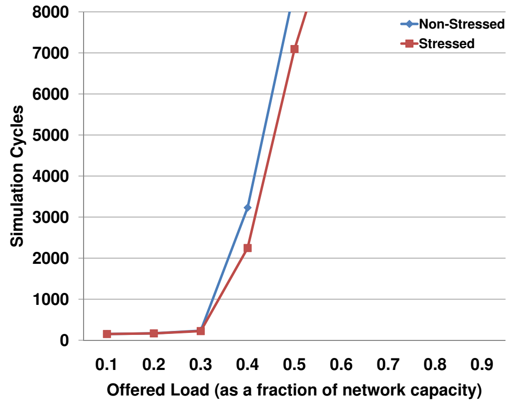


Fig. 6. Latency Comparison with Non-Stressed and Stressed Cases

Fig. 7 displays core failure percentage with regard to time. The x-axis represents the time in terms of year and y-axis represents the percentage of core failure. The red solid line represents our new approach ("New") while the blue dash line represents the one without our strategy ("Old"). The difference in terms of yield becomes obvious after 2 years and starts to become diverse. We used Monte-Carlo simulation in

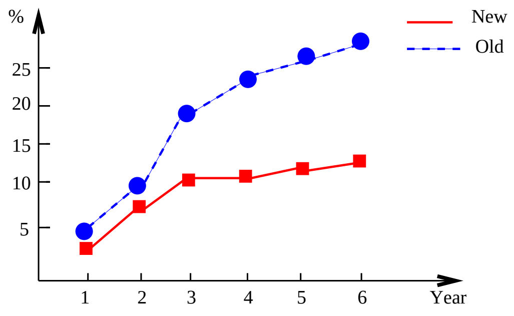


Fig. 7. Core Failure Percentage Comparison Between New strategy and without New Strategy

HSPICE to monitor the critical path delay and total leakage power for each core, and predicted the changes in MTTF. The MTTF computation models come from [16] which includes a series of variety of MTTF estimation approaches with regard to the core activity in term of workload. Fig. 8 shows the MTTF comparison between multi-core system without the proposed methodology ("Old") and with the proposed methology ("New"). The x-axis presents the time in terms of years of operation. The y-axis is the MTTF result that shows the average MTTF of a 8/64 multi-core system. Though after about 3 years both cases observe decreases in MTTF. The "New" one shows about 30% less changes.

V. Conclusion

This paper presents a new design framework for multi-core system to include device wear-out impact. The new approach started from device fractional NBTI model, provides a new NBTI aware system workload model, and a new DTP algorithm to balance workload and traffic load among active cores while relaxing stressed ones.

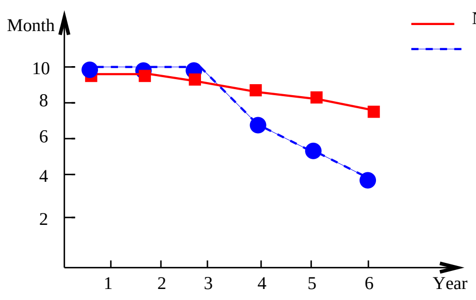


Fig. 8. MTTF Comparison Between New strategy and without New Strategy

References

[1] K. Constantinides, S. Plaza, J. Blome, V. Bertacco, S. Mahlke, T. Austin, B. Zhang, and M. Orshansky, "Architecting a Reliable CMP Switch Architecture", *ACM Trans. on Architecture and Code Optimization*, Vol. 4, no. 1, March 2007.

[2] W. Wang, S. Yang, S. Bhardwaj, R. Wattikonda, S. Vrudhula, F. Liu, Y. Cao, "The Impact of NBTI on the Performance of Combinational and Sequential Circuits", in *Proc. DAC*, 2007, pp. 364-369.

[3] S. Bhardwaj, W. Wang, R. Vttikonda, Y. Cao, S. Vrudhula, "Predictive Modeling of the NBTI Effect for Reliable Design", in *Proc. CICC*, 2006, pp. 189-192.

[4] M. Mani, A. Devgan, and M. Orshansky, "An Efficient Algorithm for Statistical Minimization of Total Power under Timing Yield Constraints", in *Proc. DAC*, 2005, pp. 309-314.

[5] E. Karl, P. Singh, D. Blaauw, D. Sylvester, "Compact in situ Sensors for Monitoring NBTI and Oxide Degradation", IEEE International Solid-State Circuits Conference (ISSCC), February 2008

[6] H. Chang and S. S. Sapatnekar, "Statistical timing analysis considering spatial correlations using a single PERT-like traversal", in *Proc. ICCAD*, 2007, pp. 621-625.

[7] H. Jiang and C. Dovrolis, "Source-level IP packet bursts: causes and effects", in *Proc. IMC*, 2003, pp. 301-306.

[8] W. Shi, H. Macgregor and P. Gburzynski, "A scalable load balancer for forwarding internet traffic: exploiting flow-level business", in *Proc. ANCS*, 2005, pp. 145-152.

[9] W. J. Dally and B. P. Towles, *Principles and Practices of Interconnection Networks*, Morgan Kaufmann, 2003.

[10] B. Parhami, *Introduction to Parallel Processing Algorithms and Architectures.* Kluwer Academic, 2002.

[11] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing.* McGraw-Hill Education, 1986.

[12] T. C. Hu, "Parallel sequencing and assembly line problems", *Operations Research*, vol. 9, no. 6, pp. 841-848, Nov.-Dec., 1961.

[13] H. El-Rewini and M. Abd-El-Barr, *Advanced Computer Architecture and Parallel Processing.* John Wiley & Sons, Inc., 2005.

[14] R. R. Muntz, and E. G. Coffman Jr., *IEEE Trans. on Computers*, Vol. C-18, no. 11, November 1969.

[15] T. Schonwald, J. Zimmermann, O. Bringmann and W. Rosenstiel, "Fully Adaptive Fault-Tolerant Routing Algorithm for Network-on-Chip Architectures", in *10th Euromicro Conference on Digital Systgem Design Architectures, Methods and Tools*, 2007, pp. 527-534.

[16] Y.Liu, W. Tang, and R. Zhang, "Reliability and Mean Time to Failure of Unrepairable Systems With Fuzzy Random Lifetimes", IEEE Trans.on Fuzzy Systems, Vol. 15, No. 5, Oct. 2007.