Frequency Scheduling For Resilient Chip Multi-Processors Operating at Near Threshold Voltage

Ying Wang, Huawei Li, Xiaowei Li
State Key Laboratory of Computer Architecture,
Institute of Computing Technology, Chinese Academy of Sciences
{wangying2009, lihuawei,lxw}@ict.ac.cn

***Abstract*—With the recently proposed redundancy-based core salvaging technology, resilient processors can survive the threat of severe timing violation induced by near-threshold *Vdd* and function correctly at aggressive clock rates. In our observation, proactively disabling the weakest components that limit the core frequency can still maintain a higher throughput at Near Threshold Voltage (NTV) supply if the cores with defected components are salvaged at a low cost. In this work, a resilience-aware frequency scaling and mapping strategy that considers defected processor states in scheduling is proposed to exploit the fault-tolerant architectures for higher energy efficiency. In our evaluation, it is witnessed that typical resilient multi-core processors can achieve significantly higher performance *per* watt in experiments compared to conventional scheduling policy.**

Keywords—NTV; multi-core; scheduling;fault-tolerant

I. INTRODUCTION

There are generally two technical routes proposed to combat the power wall that constrains the performance scaling of modern multi-core processors. One trend is to increase the proportion of "dark silicon" on-chip to keep the power consumption at bay [1] [2]. For example, in power-constrained chip multi-processors (CMP), some cores are turned-off (become dark) to allow the others switch at a full clock rate. In contrast, rather than deactivating transistors/cores as dark silicon, "dim silicon" chooses to exploit near-threshold computing (NTC) for power capping since dropping voltage leads to almost cubic power reduction.

Fig. 1 depicts the scenarios of both dark silicon and dim silicon. The first one is gating off some cores as "dark silicon" and using the remaining power to boost other cores in the normal-voltage island. The second one is dropping the voltage to near-threshold level and throttling down the frequency simultaneously, so that it renders more "dim" cores operating at a safe frequency to avoid fatal timing faults in logics. As shown in Fig. 1, a power-constrained multi-core processor must be tuned to pursue either single-core performance or core availability as the two extremities of spectrum. However, according to our observation, the operating space has some unexplored regions that might offer the best of both dark silicon and dim silicon. In this work, we leverage the state-of-the-art fault tolerant architectures to explore such *pareto* optimum space through dim silicon and dark silicon combination.

In NTC processors, the majority of dim silicon does not necessarily fall victim to the timing faults caused by the lowered V_{th} when the cores operate at full frequency. Because of the process variation, it is thought that only a limited set of bottleneck "weak" logics fails to meet the timing constraint of normal frequency. If we can deal with the "weak" logics gracefully in NTC, both single-core performance and core-level parallelism can be preserved.

In this work, we attempt to find a new way of maintaining both "core availability" and "core performance" at the same time within the power cap, by resorting to the emerging resilient CMPs with intra/inter-core salvaging support. Compared to conventional dark silicon approaches, deactivating "weak silicon" sacrifices only the fine-grained "incompetent" components created by process variation and near-threshold voltage, so that it keeps the rest of the core running at full scale and loses only the least opportunity cost of silicon area to exploit energy efficiency. It is concluded that near-threshold voltage in nanotechnologies creates more dramatic fluctuation in the frequency of sequential circuits than normal voltage supply [3]. Directly over-clocking the cores leads to timing faults in certain logics wrecked by worst-case timing variation, which is forbidden for reliability purpose. However, adopting the frequency decided by the worst-case critical path leads to power inefficiency due to under-clocking. In fact, most of the components can still operate safely at a higher clock rate if only the timing defects in the "weakest" components are tolerated by protection techniques. Recently proposed core salvaging techniques provides a viable approach to transform the "weakest silicon" into "dark silicon" for higher energy efficiency. It is because the timing failures of a particular component is not always detrimental to processors, and could be tolerated by salvaging techniques from all layers [4] [5]. Such resilient architectures often resort to inexpensive intra-core instruction migration or instruction translation to tolerate the failure of particular components.

For example in Fig. 1 (c), the CMP is able to work under NTV without compromising the core speed. It survives the PV-induced timing failures in weak silicon by "outsourcing" the functions to other cores through the salvaging logics. In this case, it leverages NTV for power reduction but trades off the weak components for higher core performance, so that only the weakest part of silicon becomes "dark".

However, two things are very critical for exploiting the weak silicon for energy efficiency: First, the optimal core frequency assignment is hard to find. Traditional frequency scaling algorithm only concerns the program characteristics (memory intensive or compute intensive) and power constraint to decide the best core state (p-state). However, the frequency setting of resilient architectures impacts the distribution of weak silicon in cores and indirectly influences the workload by changing the core performance and the cost of core salvaging. The second one is the mapping policy. Due to process variation, cores exhibit different timing behavior and create varied amount of weak silicon at different operating frequencies. How to match the threads onto

The work was supported by National Natural Science Foundation of China under Grant 61432017, Grant 61504153, Grant 61532017 and Grant 61221062.

cores is critical to the overhead of fault tolerance. For example, a thread does not use the FPU so that it can run smoothly on the core with a weak FPU fails at a high clock rate, so it could be mapped to that core with little fault tolerant overhead.

In this work, we investigate what is the best frequency scaling and thread mapping strategy in resilient multi-core processors where the constraint of reliability is relaxed and logics are exposed to the timing defects generated by NTC.

II. Methodology

A. Target architecture

Fig. 2 depicts a 16-core chip multiprocessor. The detailed parameters and configuration of such a CMP is described in Table-I. Because the per-core voltage scaling entails unmaintainable overhead of voltage regulator and power delivery network [6] in multi-core processors, we assume that all cores share a common voltage domain, whilst NoC and the cache banks share another independent voltage domain. Each core has an independent clock domain. Due to the impacts of process variation, each individual core has a random maximum fault-free operating frequency and also a unique set of fault maps associated to different clock rates. When component failures occur, the CMP migrate the inexecutable instructions to a closely-coupled core through salvaging fabrics [5].

B. The overall framework

It includes two key steps to exploit weak silicon to gain power efficiency in NTC CMPs as it illustrates in Fig. 2. The ***first*** step is to use off-line testing to profile the fault maps of each core under a set of clock rates supported. Each fault map together with its associated clock rate corresponds to a *p-state* of the core. Because the timing defects indirectly influence the performance of the core through salvaging technique, the *K* fault maps under *K* clock rates for each core will be used in the performance predictor to help make scheduling decision. The ***second*** step is to obtain the critical parameters of the running workloads, which are also needed in the on-line performance predictor. The workloads profile obtained by performance monitors includes several key parameters that impact the performance of cores operating at a certain *p-state*. As it is shown in Fig. 2, the performance predictor combines the off-line fault maps and the on-line workload profile to estimate the system performance and power consumption in different frequency/mapping setting, and then finally relies on the proposed scheduling algorithm to search for the near-optimal operating points of all cores.

C. Modeling processor state (p-state) through off-line test

In cores with core salvaging techniques, a salvageable component is associated with a cluster of instructions that relies on it, which is defined as *instruction cluster*. The instructions needed to be migrated or re-translated are deemed as *offending instructions*. For example, the failure of SIMD accelerator is associated to the instruction cluster of SSEx.

Modeling the new *p-states* has to be fully aware of timing variation distribution in each core and needs the fault maps generated by off-line timing test as discussed in last section. For a CMP under timing test, we pre-obtain the timing fault map of each core under all supported clock setting $f_0, f_1, \dots, f_K$, and convert it into a ISA-integrity map IM:

$$IM = \{\mathrm{IM}_i | 0 < i < C\},$$
$$\mathrm{IM}_i = \{IM_i^0, IM_i^1, \dots, IM_i^K\},$$
$$IM_i^k = max(IM_{i,0}^k, IM_{i,1}^k, \dots, IM_{i,L}^k) \quad (1)$$

where C is the number of instruction clusters that are related to a certain hardware unit in core. L is the number of salvageable hardware components within the core. $IM_{i,j}^k$ denotes the migration overhead of instructions belonging cluster-i when component-j fails under frequency-k, given that instruction migration is used to salvage the component. $IM_{i,j}^k$ equals 0 if all components related to that instruction cluster are safe. IM_i^k only stores the maximum salvaging penalty if there are multiple failure sources that triggers the salvaging mechanism. Besides the ISA-integrity maps, the fault map CM is also available after test. CM is a $L \times K$ binary fault map obtained in test. CM_i^j indicates that component-i failed at clock rate f_j.With the ISA-integrity and fault map obtained via off-line test, we deduce the instruction salvaging overhead and use it to predict the performance under a certain *p-state*.

TABLE I. Baseline CMP configuration

Cores	OoO, two-issue, x86, 1.5/1.2/1.0/0.8GHz
Networks	4×4,2D-Mesh, One common inter-core salvaging queue for each 2×2 sub-mesh
Technology	22nm node
L1 DCache	32KB, 4 way, 2 cycles hit
L2 Cache	Private, Exclusive, 8 way, 64B lines, 16 private banks, 1MB/bank, LRU replacement
Memory	4G DRAM, 260 cycles access
Coherence	MOESI directory

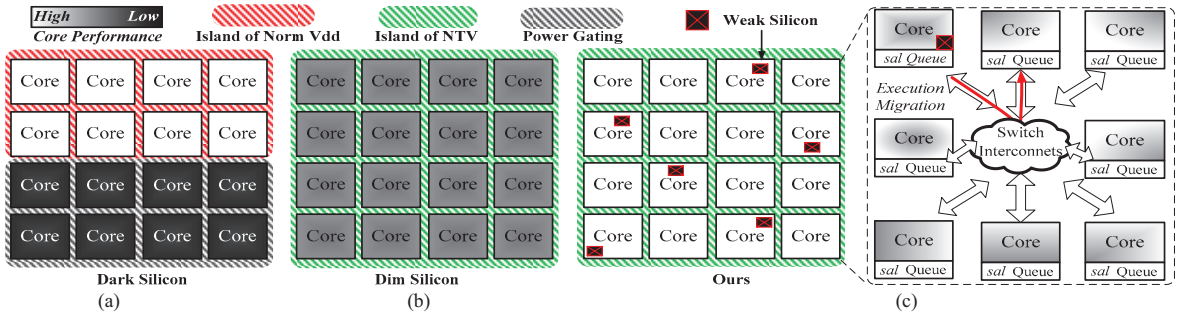

Fig.1 Three scenario to dispose "silicon" in power-hungry CMP chips: (a) Dark silicon: Core-level utilization wall (b) Dark silicon: Performance throttling (c) Exploiting weak silicon through redundancy-based salvaging

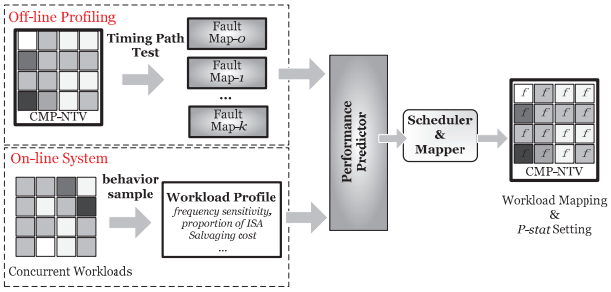

Fig. 2 The framework of weak silicon: resilience-aware DFS and mapping policy

D. Modeling the performance of cores with weak silicon

The execution time of application is composed of computation phase $T_{compute}$ and memory phase T_{mem}. The objective of application CMP scheduling is to minimize the execution time *(T)*.

We follow the model of frequency tuning for realistic CMP systems [7] to predict workload execution time, except that the factor of prefetching stall is excluded in equation-(2). T_{memory}^{min} is the minimum time the DRAM system needs to satisfy all of the memory requests, which is a bound of memory bandwidth.

$$T = max\{T_{memory}^{min}, T_{compute} + T_{mem}\}$$
$$T_{compute} = (IC \times T_{ins} \times ILP)/f$$
$$IC = \sum_{i=0}^{C-1} IP_i \quad (2)$$

IC indicates the total instruction count of workloads, f is the core frequency, ILP is instruction level parallelism, and T_{ins} is the average execution time of a single instruction, which is thought as quasi-linear to core frequency f_j. IP_i records how many instructions in workload belongs to the instruction cluster-*i*.

$$\frac{T_{compute}}{ILP \times IC} = \frac{(IC \times T_{ins})}{f_j \times IC}$$
$$= \sum_i^{C-1}\left(IP_i \times T_{home} \times \left(1 - CM_i^j\right)\right)$$
$$+ \sum_i^{C-1}(IP_i \times T_{foreign} \times IM_i^j) \quad (3)$$

According to equation-(3), $T_{compute}$ includes two parts of overhead: the local execution time, and the outsourcing overhead, i.e. $ILP \times IC \times \sum_i^{C-1}(IP_i \times T_{foreign} \times IM_i^j) = F(f_j)$. T_{home} is the timing overhead of executing the instruction in the original core while $T_{foreign}$ indicates the overhead of instruction migration. In total, this equation implies that the performance of thread does not only depend on the frequency of the assigned processing core, but also the ISA dis-integrity of that core, i.e. the cluster of instructions that fail to run in the core. Therefore, before thread mapping and frequency scaling, we should pay attention to the code composition and the supported instruction clusters of all cores under all range of frequencies.

E. Resilience-aware frequency scaling and mapping

We formulate the problem of resilience-aware frequency scaling and mapping for an *N*-core processor as:

$$Obj: minimize\{\sum_0^{N-1} T_{compute}^i\}$$
$$s.t.:$$
$$P = \sum_{i=0}^{N}(f_i \times Vdd_i + l_i) < P_{cap} \quad (4)$$

Power P is restricted by the power cap P_{cap}. l_i is the constant leakage power depending on Vdd, temperature and Vth. Where $T_{compute}^i$ of application-*i* executed on core-*j* at frequency f_i is modeled in the form according to equation-(3):

$$T_{compute}^i = a_i/f_i + F(f_i) + O(M_{i,j}) \quad (5)$$

Searching the space of f_i and threads mapping for optimal operating point has to explore a solution space as huge as $K^N \times$ N! for an *N*-core processor. In formula-(5), a_i is a coefficient decided by *ILP* and memory behavior of application-*i* as described in formula-(2) and formula-(3). All these factors could be profiled in monitor phase, and assumed to be fixed in a sufficiently-short scheduling interval. $F(f_i)$ indicates the function of outsourcing overhead to f_i according to equation-(3), decided by the proportion of out-sourced instructions. Suppose M is the assignment matrix, $M_{i,j}$ indicates the app-*i* is mapped to core-*j* if it equals one, otherwise it is reset to null. $O(M_{i,j})$ is the outsourcing overhead induced by resource contention between the offending instructions and the native instructions in the destination core. The factor depends on the proportionality of outsourced instructions in the offending app-*i* and that of the same instructions in the native app-*l*.

$$O(M_{i,j}) = r\sum_{p=0}^{C}\left(\left(IP_{i,p}/IC_i + IP_{l,p}/IC_l\right) \times IM_p^{j,k}\right) \quad (6)$$

r is the contention coefficient. $IM_p^{j,k}$ is the ISA-integrity map of core-*j*. $IP_{l,q}$ *and* IC_l is the instruction proportion of the native thread app-*l* in the salvaging core paired to core-*j*. With the energy-performance model, it still entails two key steps to achieve the ideal thread scheduling.

Step-1: profile the key factors of performance model

Where $T_{compute}^i$ of application-*i* is modeled in the form of a_i/f_i+$F(f_i)$+$O(M_{i,j})$. To predict $T_{compute}^i$ of a certain frequency setting, we must also search the fault map IM_i^j of each core to get the value $F(f_i)$ for every frequency knob according to equation-(3). Therefore, behavior sampling is needed to predict the throughput according to equation-(2) and equation-(3). For example, in sampling phase, threads are randomly mapped onto *N* defect-free cores that are running at a conservative clock rate. In this phase, *ILP,* IP_i and T_{ins} are monitored through the performance counters to derive a_i for app-*i*. IP_i records the proportionality of instruction clusters executed by app-*i*, which is used to derive b_i and $O(M_{i,j})$ together with the fault maps for a certain assignment. Thus, we only need to schedule the threads onto the initialized cores and do sampling for one time to record the proportionality of instruction clusters: $\{IP_0, IP_1, \ldots, IP_m\}_i$. For each app-*i,* a_i and *O(M)* factors are then fitted into formula-(5) to estimate the workload execution time.

Step-2: Search for the optimal mapping and *f*-setting

We start from the common approaches and expect to find effective heuristics to resolve the problem-(4).

Greedy Algorithm The intuition about greedy scaling for the maximum throughput is to assign the individual thread that yields the maximum margin utility from increased clock rate to the core with has the highest and defect-free clock rate:

1. First, rank coefficient a of all threads in ascending order

2. Then rank the core in ascending order of their highest defect-free frequency.
3. Finally, map the threads to cores one by one.

Tentative global mapping: Tentative global mapping firstly tries to overclock each core until at least one component becomes unstable, then runs the assignment procedure with Hungarian algorithm to get the optimal assignment matrix *M*, then triggers another round of overclocking until at least one more component fails. Please note that "overclocking" here does not need to increase the frequency of the real cores and do sampling afterwards, it only mimics the step by looking up the fault maps of all cores, and uses the profiled factors to predict the $T^{i}_{compute}$ using formula-(3). If the summed throughput outperforms the old mapping, the new assignment will be accepted as the final mapping. Otherwise, the new mapping is tentatively accepted. However, any core experiencing performance degradation with the new *p-state* will be throttled down to the old state. For this tentative mapping, another round of reassignment procedure is triggered. If the final throughput still lags behind that of old mapping, the tentative mapping will be rejected, and the old mapping will be marked stable as the final configuration. The complexity of such algorithm is $O(n^3)$. At the same time, if power cap is violated in the procedure, the decision is also rejected.

III. Evaluation

A. Experimental Setups

Baseline CMP System-level evaluation is performed using Sniper-6.0 [8]. The detailed parameters of chip multiprocessor can be found in Table I.

Instruction-level salvaging The simulated CMP supports instruction-level component salvaging as described in [5] [9]. In this architecture, every four adjacent four cores shared an inter-core queue that migrates the instruction from defect core to normal core when the offending instructions are detected in defect core.

Timing Variation VARIUS-NTV is used to derive the PV maps and the timing information of components for each core [3].

Workloads description Multiple-programmed workloads mixed of SPEC2006 benchmarks are selected to mimic the application of CMP in datacenter or cloud. The benchmarks from SPEC2006 suites are partitioned into two groups as two multi-programmed workloads. For each group, the benchmark threads are respectively bounded onto of the 16 cores after assignment, so that there are always multiple different programs running on the CMP.

In experiment, *baseline* is a variation-aware scheduling algorithm that uses a linear programming solver to decide the core frequency [10]. In *baseline*, the NTC cores always run at a fault-free state. The performance and power of all other schemes are normalized to *baseline* for comparison. *Greedy* and *tentative global mapping* are all evaluated and compared.

Fig. 3 shows the performance (measured in the average execution time of threads) of the two workloads running on 16-core CMP. The results are normalized to *baseline* running at maximum allowable defect-free frequency at NTV supply. It is shown that tentative global mapping with weak silicon completes the threads 16% faster than baseline within the same power budget. The performance of greedy is only 6% better than baseline. Comparatively, dark silicon cannot afford 16 simultaneously-running cores, so that it takes longer time to complete the whole workload than weak silicon though it completes each thread faster.

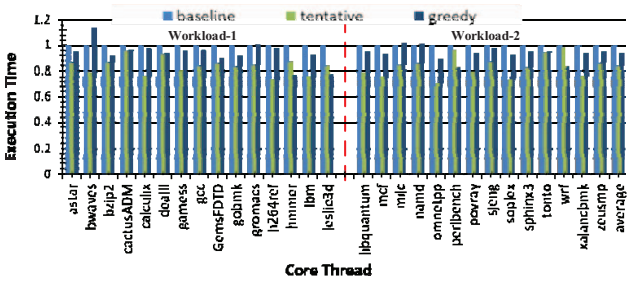

Fig. 3 Performance comparison: weak silicon vs dim silicon

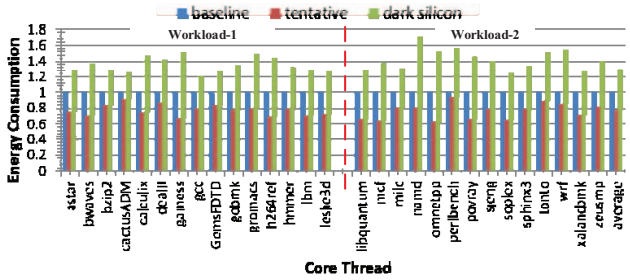

Fig. 4 Energy comparison: dim silicon vs weak silicon vs dark silicon

Fig. 4 shows the energy comparison results. Tentative global search consumes 78% of the total energy it takes for *baseline* to complete the two workloads on average. In Fig. 4, dark silicon indicates the total energy it takes for the CMP operating at normal voltage supply. Though dark silicon can complete each single thread quickly by boosting the core with higher *Vdd*, it takes a longer time and more energy to complete the whole combination of multiple threads because dark silicon turns off cores to maintain a high frequency for the remaining cores due to the power cap.

IV. Conclusion

This study is the first work that proactively creates defected processor states to accelerate the application at NTV provision by utilizing emerging instruction-level core salvaging techniques. With the help of resilience-aware frequency scheduling and mapping algorithm, weak silicon is able to expand the space of energy-efficient operating points for CMPs with limited power budget, leading to a considerable energy-efficiency and performance improvement over the conventional PV-aware frequency scheduling policy in the fault-tolerant CMPs.

References

[1] G. Venkatesh, *et al.*, "Conservation Cores: Reducing the Energy of Mature Computations," In *Proc. ASPLOS*, 2010.

[2] H. Esmaeilzadeh, *et al.*, "Dark Silicon and the End of Multicore Scaling," in *Proc. ISCA*, 2011.

[3] U. R. Karpuzcu, *et al.*, "VARIUS-NTV: A microarchitectural model to capture the increased sensitivity of manycores to process variations at near-threshold voltages," In *Proc. DSN,* 2012.

[4] M. D. Powell, et al., "Architectural Core Salvaging in a Multi-Core Processor for Hard-Error Tolerance," in *Proc. ISCA*, 2009.

[5] A. Pan, et al., "Improving Yield and Reliability of Chip Multiprocessors," in Proc. DATE, 2009.

[6] K.Wonyoung, *et al.*, "System level analysis of fast, per-core DVFS using on-chip switching regulators," in *Proc. HPCA*, 2008.

[7] R. Miftakhutdinov, *et al.*, "Predicting Performance Impact of DVFS for Realistic Memory Systems", in *Proc. MICRO*, 2012.

[8] W. Heirman et al., "Power-Aware Multi-Core Simulation for Early Design Stage Hardware/Software Co-Optimization," in *Proc. PACT*, 2012.

[9] S. Srinivasan, *et al.*, "A Runtime Support Mechanism for Fast Mode Switching of a Self-Morphing Core for Power Efficiency," in *Proc. PACT*, 2014.

[10] R. Teodorescu, et al., "Variation-Aware Application Scheduling and Power Management for Chip Multiprocessors," in *Proc. ISCA*, 2008.