Power Efficient Redundant Execution for Chip Multiprocessors

Pramod Subramanyan, Virendra Singh
Computer Design and Test Lab,
Supercomputer Education and Research Center,
Indian Institute of Science, Bangalore.

Kewal K. Saluja
Electrical and Computer Engineering Dept.,
University of Wisconsin-Madison,
Madison, WI

Erik Larsson
Dept. of Computer and Info. Science,
Linkoping University,
Linkoping, Sweden.

***Abstract*—This paper describes the design of a power efficient microarchitecture for transient fault detection in chip multiprocessors (CMPs) We introduce a new per-core dynamic voltage and frequency scaling (DVFS) algorithm for our architecture that significantly reduces power dissipation for redundant execution with a minimal performance overhead. Using cycle accurate simulation combined with a simple first order power model, we estimate that our architecture reduces dynamic power dissipation in the redundant core by an mean value of 79% and a maximum of 85% with an associated mean performance overhead of only 1.2%.**

I. Introduction

Increasing levels of integration, reduced supply voltages and higher frequencies are causing soft errors to become a significant problem for high performance microprocessors [25], creating a need for high performance, low power and fault tolerant microarchitectures. While a low performance overhead due to fault tolerance is an obvious requirement, a low power overhead is also just as important because power dissipation and peak temperature are one of the key performance limiters for modern processors [5]. Reducing power dissipation also has the additional advantage of reducing operating temperatures, which can significantly increase chip reliability [18].

One set of approaches to tackle the soft error problem attempt to deduce the occurrence of an error by monitoring and identifying deviations or perturbations in program behavior [20, 30]. Another set of approaches employ some form of redundant execution coupled with input replication and output comparison [1, 11, 16, 21, 22, 26]. Architectures like Restore [30] and Perturbation Based Fault Screening [20] are attractive for applications that do not require comprehensive transient fault coverage as they impose only a small overhead in terms of performance and power. Conversely, architectures based on redundant execution typically have a larger performance and power penalty but usually provide higher transient fault coverage. An example of this kind are architectures based on redundant multithreading (RMT) which have a performance overhead that varies between 20-30% [4, 11, 16, 21, 22].

While RMT remains an attractive option for transient fault tolerance, the base assumption made in RMT designs is a simultaneously multithreaded processor. However, SMT processors require a wide and deep superscalar pipeline to achieve optimal power-performance[12], which lead to large layout blocks and additional circuit delays [17]. In some cases, destructive interference between threads executing on an SMT processor can negate the performance advantages of SMT [14]. Therefore, we believe fault tolerant microarchitectures with small performance and power penalties which are not based on RMT need to be studied.

Our architecture designates one of the cores of a CMP as the primary-core (P-core), and the other core as a redundant core (R-core). Branch outcomes, load values and fingerprints are transferred over a dedicated interconnect between the P-core and the R-core. Transient faults are detected by comparing fingerprints. Error correction is achieved by restoring the program state to most recent validated checkpoint.

The key innovation introduced in our architecture is dynamic frequency and voltage scaling based on the number of outstanding entries in the branch outcome and load value queues. An increasing number of outstanding entries in either one of the queues the means that the P-core is executing instructions faster than the R-core, therefore to avoid performance degradation due to the P-core filling up the queues and stalling, the frequency of the R-core is increased. On the other hand, in the common case, the number of outstanding entries in the R-core is quite small. This is because in our architecture the R-core does not misspeculate or access the data cache - obtaining data from the P-core instead (see Section III), and hence it can operate at a lower frequency than the P-core with only a small amount of performance degradation.

The rest of this paper is structured as follows. Section II describes related work. Sections III and IV describe the design and tackle some implementation issues. Section V evaluates the design: Section V-A describes our simulation methodology and Section V-C presents the results of our evaluation. Finally section VI concludes.

II. Related Work

Fault Tolerant Architectures: Todd Austin introduced DIVA [1] which is a novel fault detection and correction architecture which uses an in-order *checker* processor to detect errors in a larger out-of-order superscalar processor core. The checker processor is fabricated using larger and more reliable transistors. Although using larger transistors makes DIVA less susceptible to soft errors, DIVA cannot guarantee the detection of all soft errors, especially those that occur in the checker processor. In the Selective Series Duplex (SSD) architecture [8], the main superscalar processor pipeline is duplicated selectively to form the V-pipeline. The V-pipeline *verifies* the operation of main CPU-pipeline by re-executing non-speculative instructions. One disadvantage of the SSD architecture is that resources and functional units allocated for re-execution are unavailable for normal execution (*i.e.*, when redundant execution is not being performed).

Transient fault detection using simultaneous multithreading was introduced by Rotenberg in AR-SMT [22] and Reinhardt and Mukherjee in SRT [21]. A simultaneous and redundantly threaded (SRT) processor augments SMT processors with additional architectural structures like the branch outcome queue and load value queue to enable transient fault detection. The branch outcome queue enhances the performance of the redundant thread, while the load value queue provides input replication. Muhkerjee et al. also introduced chip level redundant threading (CRT) [16], which extends SRT to simultaneously multithreaded chip multiprocessors. Gomaa et al. studied Chip Level Redundant Threading with Recovery (CRTR) [4], which uses the state of the trailing thread to recover from an error. Since the primary and redundant threads are executed on different cores in CRT and CRTR, they reduce the amount of data transmitted over the

interconnect between cores by the method of Dead and Dependence Based Checking Elision (DDBCE).

A number of variants based on SRT processors have been proposed. An example is Speculative Instruction Validation (SpecIV) [11] which reduces the performance overhead of SRT by predicting the expected values of instruction results and re-executing only those instructions whose results differ from the expected values. By reducing the number of instructions re-executed, SpecIV will also reduce the power overhead of the redundant thread. Note that SpecIV fetches and decodes all instructions even if they are not re-executed in the redundant thread, decreasing potential power savings.

DIVA, SSD, SRT and its variants all use some form of redundant execution to detect the occurrence of an error. An alternate approach introduced by Razor [2] is circuit level augmentation of a processor design to detect transient errors. Razor replicates critical pipeline registers and detect transient errors by comparing the value stored in the two registers. A similar architecture is SPRIT^{3}E[28], which improves performance by operating by the processor a clock frequency that is higher than the worst-case frequency. The SPRIT^{3}E architecture also augments all time-critical pipeline registers with a *backup register*. The clock input to the backup register is a phase-shifted version of the main clock. Comparison of the main and backup registers can detect timing errors. SPRIT^{3}E is able to increase significantly increase the performance of a superscalar processor but this increase comes at the cost of additional area and power.

While schemes based on RMT are attractive as they provide complete transient fault coverage with only small performance and power penalty, the base assumption made in RMT designs is a simultaneously multithreaded processor. Lee and Brooks [12] studied power-performance efficiency for CMP and SMT processors and found that efficient SMT processors require wider (*e.g.*, 8-way) and deeper pipelines. Since many microarchitectural structures scale in a non-linear fashion with increasing issue width [17], wider pipelines require larger area and lead to lower clock rates. A study by Sasanka et al. [23] found that CMPs outperform SMT architectures for multimedia workloads. Another study by Li et al. [13] found that CPU bound workloads perform better on CMP architectures while SMT architectures perform better on memory bound workloads. A further problem with SMT architectures is the destructive interference among threads that sometimes causes performance degradation [14].

The above results indicate that alternative architecture that can provide transient tolerance with performance/power overheads similar to those of SRT or its derivatives might be appealing. Hence, our work targets improving power-efficiency of redundant execution schemes on CMPs using an architecture that is *not* based on SMT processors.

Smolens et al. [27] introduced fingerprinting, which reduces the bandwidth required for state comparison. Fingerprinting summarizes the execution history and current state of a processor using a hash value. Transient faults are detected by differences in the hash value computed by the two cores. A related architecture is Reunion [26] which provides input replication in chip multiprocessors without the requirement of lockstepped execution by reusing the soft error handling mechanisms for dealing with input incoherence. Reunion requires changes to cache coherence controller which is a component that is difficult to design and verify.

Dynamic Voltage and Frequency Scaling (DVFS): Isci et al. [5] introduced a set of policies to manage per-core voltage and power levels in CMPs. Their policies aim to maximize performance while keeping power dissipation under the power budget. These are managed either by a dedicated micro-controller, or a daemon running on a dedicated core. In contrast, our design utilizes occupancy information of the structures added for redundant execution to manage the power level of the redundant core without software intervention and very little additional hardware. Kim et al. [9] described the detailed design of on-chip regulators showing that it is possible to perform voltage changes in time periods of the order of a few hundred nanoseconds. Although current commercial processors do not yet have the ability to set per-core voltage levels, the AMD Quad Core Opteron [6] allows the frequency of each core to be set independently.

Reducing Energy Consumption in Fault Tolerant Architectures: Montensinos et al. [15] use register lifetime prediction to provide ECC protection to a small subset of the physical register file. Their approach reduces the power dissipation of the register file at the cost of additional area for the ECC table. To the best of our knowledge, this is the only work that attempts to decrease power consumption in the context of transient fault detection.

III. Proposed Design

A. Base Architecture

Our architecture designates one of the cores of the CMP as a *primary core (P-core)*, and another core as the *redundant core (R-core)*. The primary and secondary core execute the same instruction stream, with the same input data stream, but the P-core is temporally "ahead" of the R-core, *i.e.,* the R-core executes a particular instructions after it has been executed by the P-core. To enable power efficient redundant execution, we augment the core with some additional structures. A schematic of the modified core is shown in Figure 1. If redundant execution is disabled, then the additional structures are not used and core operates like a normal superscalar processor. The rest of this section describes the architecture in detail.

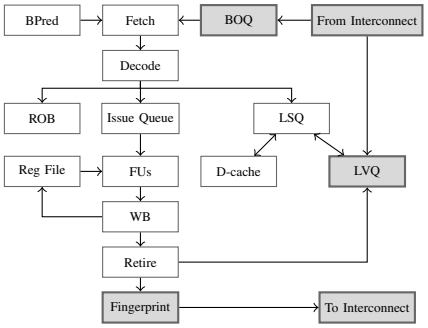


Fig. 1. Schematic diagram showing core augmented with structures required for redundant execution. Newly added structures are shaded and not used when redundant execution is not being performed.

Components Protected: Our mechanism detects errors that occur within the fetch, decode and execution units, and the register file. We assume that data cache, L2 cache and main memory are protected by ECC.

Input Replication: Input replication is the mechanism to ensure that the two cores see exactly the same input. Our architecture achieves this by using the load value queue (LVQ) structure introduced by Reinhardt and Mukherjee [21]. Loads from the P-core are transferred over the interconnect and stored in the LVQ structure in the R-core. Load values generated by instructions on wrong paths should not be transferred over the interconnect, so values are transferred over the interconnect only when the corresponding load instruction retires.

Load instructions executing in the R-core do not access the data cache and instead obtain the load values from the LVQ. Note that

since the R-core never accesses the data cache, the data cache can be completely shut down to reduce leakage power.

Although load values enter the LVQ in program order, load instructions in the R-core may be issued out of program order. Two solutions have been proposed to this problem: in [21] the authors restrict load instructions to execute in program order, while in [16] load instructions are associated with a tag generated by the primary thread and this tag is used to select the value to be loaded from the LVQ. Our solution is conceptually similar to the one in [16]. However, since the P-core and R-core are spatially separated, generating tags when the load value retires in the P-core will increase interconnect traffic. Instead we assign tags to load values as they enter the load value queue in the R-core. The tag is nothing but the address of the location in the LVQ in which the value is stored. As load instructions are decoded in the R-core, they are also assigned a tag; this tag is carried along with the load instruction and is used when the load instruction issued to read from the LVQ[1]. The above scheme works because there is a one-to-one correspondence between load instructions as they retire in the P-core and load instructions as they are decoded in the R-core.

Entries are deleted from the LVQ only when the corresponding load instruction retires. If the LVQ becomes full, then the P-core is unable to retire instructions. If the LVQ is empty, and a load instruction is issued then that load is tracked in a MSHR (Miss Status Holding Register) structure and its value returned when the corresponding load arrives from the P-core.

Information about external interrupts also needs to be transferred from the P-core to the R-core over the interconnect to ensure precise input replication. We also assume that TLB misses are handled in software, and that the instructions which are used to read the page table are also replicated like other load instructions.

Output Comparison: After the execution of every N instructions the P-core and R-core compute a hash value that summarizes updates that have been made to the state of a processor. (N is the referred to as the checkpointing interval). This hash value is referred to as a *fingerprint* [27]. The two cores swap and compare fingerprints to detect soft errors. If no soft error has occurred, the architectural updates will be exactly the same, guaranteeing that the fingerprints will also be equal. If a soft error occurs, the fingerprints are extremely likely to be different. A mismatch in fingerprints necessarily indicates the presence of a soft error.

As fingerprints capture updates to the *architectural state* of the processor, they have to be computed after instruction retirement. In addition to comparing fingerprints at the end of each checkpointing interval, fingerprints are also compared before the execution of any I/O operation or uncached load/store operation as these may have side-effects outside the processor.

Like in [26] we assume that fingerprints capture all register updates, branch targets, load and store addresses and store values.

The frequency of fingerprint comparisons affects the performance overhead of our scheme. The P-core is stalled after the computation of the fingerprint is completed and before the corresponding fingerprint is computed and returned from the R-core, so smaller fingerprint comparison intervals lead to a greater performance overhead. Smolens et al. [27] reported that for I/O intensive workloads, I/O operations occur approximately every 50,000 instructions. Based on this result, we conservatively assume a fingerprint comparison interval of 50,000 instructions.

[1]The scheme assumes that two in-flight load instructions cannot have the same tag; this assumption is guaranteed to be valid if the size of the ROB is less than the size of the LVQ - this is true for all the LVQ sizes considered in this paper.

Core to Core Interconnect: We assume that each processor has a dedicated bidirectional interconnect with an adjacent processor, and this interconnect is used for transferring the load values and branch outcomes from the primary to the redundant core. This interconnection strategy implies that the choice of primary and redundant cores is restricted to one of four pairs in the 8-core CMP. While this does decrease scheduling flexibility, it also reduces interconnect area and power requirements.

Among the pair of cores linked by the interconnect, the choice of primary and secondary cores may be made arbitrarily if all cores can operate at the same supply voltage/frequency. However, process variations can have the effect of rendering some cores to be unable to operate at the maximum voltage-frequency level [29]. In such a situation it may be beneficial to use the lower frequency core as the redundant core.

Branch Outcome Queue: Branch outcomes from the P-core are forwarded to the R-core to prevent the R-core from misspeculating. At the time of retirement, the target address of each branch instruction is transmitted over the interconnect to the R-core. At the R-core the branch instruction is added to the branch outcome queue (BOQ). During instruction fetch, the R-core does not use the branch predictor, but instead accesses the branch outcome queue to get the direction and target address of the branch. If the branch outcome queue is empty, then instruction fetching stalls.

B. Voltage and Frequency Control

A reduction in the frequency of operation of the R-core, does not significantly affect performance in our architecture. To understand why this is so, let us analyze R-core performance using the method of interval analysis described by Eyerman et al. [3]

Eyerman et al. found that performance of superscalar processors can be analyzed by dividing time into intervals between miss events.

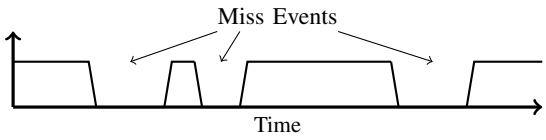


Fig. 2. Interval Analysis of Superscalar Performance. Time increases along the x-axis, and the y-axis shows the number of instructions issued.

The base assumption of the model is that superscalar processors are designed to smoothly stream instructions through the pipeline at a rate that is more or less equal to the issue width in the absence of miss events. This smooth flow instructions is interrupted by miss events like cache misses, branch mispredictions and TLB misses. The effect of miss events is to first stop the dispatch of *useful* instructions. Next, there is a period during which no useful instructions are issued, and finally when the miss event is *resolved*, the smooth flow of instructions resumes. Figure 2 depicts this interval behavior.

Miss Events in the R-core: An important observation here is that unlike the P-core where a variety of miss events can disrupt smooth instruction flow, there are only a few different miss events that affect the R-core: I-cache misses, BOQ stalls and LVQ misses. Recall that BOQ stalls occur when the target for a branch instruction is not available in the BOQ, and fetching is stalled until this outcome arrives. An LVQ miss is similar event which indicates that the R-core is attempting to obtain the value of a load instruction whose value has not arrived over the interconnect.

The key insight here is that the resolution time of the BOQ stalls and LVQ misses depends on when the corresponding instructions retire in the P-core and so does *not* change if the frequency of the R-core is reduced. Reducing the frequency of the R-core has the effect of decreasing the number of instructions issued in the R-core per unit time. This causes the length of the intervals in which useful work is performed to increase. However, *this increased interval length need not have an adverse effect on performance if the size of the interval does not increase beyond the resolution time of the next miss event*. This is depicted graphically in Figure 3.

We define the lowest frequency at which the R-core can operate without any performance degradation as its *optimal frequency*.

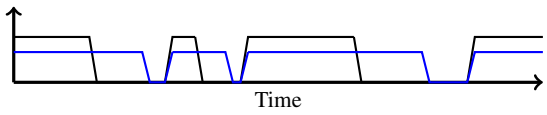


Fig. 3. Interval Analysis of the Effect of Reducing R-core Frequency. The black line shows the performance of the R-core when operating at the same frequency as the P-core. The blue line shows the performance of the R-core when operating at a reduced frequency.

Exploiting the Effects of Time-Varying Phase Behavior: It is well known that programs have time-varying phase behavior [24], causing the IPC of the P-core to vary over time. A lower IPC for the P-core has the effect of increasing the durations of the miss events in the R-core, while a higher IPC has the opposite effect. This means that when the program is executing a phase of low IPC, then the R-core can operate at low frequency, and during phases of high IPC, the R-core can increase its frequency.

The question now becomes how do we identify these phases of low/high IPC in the programs during execution. In this context we make the observation that the sizes of the BOQ and LVQ are an indication of the difference in execution speed between the P-core and the R-core. To understand why, let us assume for the moment that the R-core has infinite sized BOQ and LVQ structures. If the R-core is operating at a lower than its the optimal frequency, then it will not suffer any miss events due to BOQ stalls or LVQ misses (as it much "behind" the P-core), and the number of elements in the LVQ and BOQ will continuously increase. On the other hand, if the R-core is operating at higher than its optimal frequency, the LVQ/BOQ structures will mostly be empty, as the R-core will consume these entries very quickly after they enter the structures. This suggests that an algorithm which varies the frequency of the R-core based on the number of entries in the queues will be able to track IPC variations in P-core; reducing power dissipation without adversely affecting performance.

DVFS Algorithm: Our algorithm periodically samples the size of the BOQ and LVQ after a fixed time interval T_s. There are two thresholds associated with the BOQ and LVQ - a high threshold and a low threshold. If the occupancy of any one structure is greater than its high threshold, then the frequency of operation is increased. If the occupancy of one of the structures is less than the low threshold, then the frequency of operation is decreased. In effect the algorithm attempts to maintain the occupancy of the structures in between the low and high thresholds.

The thresholds can be set either statically or dynamically. Our results in section V-C show that a single static threshold provides significant power savings with only a small performance degradation, so in this paper we only use a single statically set threshold value. The effect of different threshold values is explored in section V-C.

IV. Discussion of Fault Recovery and Coverage

There are two ways in which faults can be detected by our design. One is through the comparison of fingerprints yielding fingerprints which do not match. In this case, control is transferred to the OS which restores execution to a previously taken checkpoint. Alternatively, since during fault-free execution the R-core always gets perfect branch predictions from the P-core, the detection of a misspeculation in the R-core indicates the occurrence of a transient fault. This is handled in the same way as a mismatch in fingerprints. A mismatch in the fingerprints calculated by the two cores necessarily implies a soft error, but a soft error may be such that the fingerprints computed at the two cores still match. This situation referred to as *fingerprint aliasing* and occurs with probability $2^{-(p-1)}$ where p is the width of the fingerprint, and hence larger fingerprints can be used to reduce the probability of aliasing to an acceptably low level.

Our design does not require the BOQ to be protected with ECC for correct operation as a soft error in the BOQ can be detected in the R-core. However, protecting the BOQ reduces the likelihood of an unnecessary recovery operation by a small amount. Whether the LVQ requires protection with ECC for correct operation depends on how the fingerprint computation is implemented. If we use the scheme suggested in [26] where fingerprints are computed using instruction results stored in the ROB, the value stored in the P-core ROB as well as values stored in the LVQ need to be protected with ECC.

We assume that caches are protected with ECC. Our design detects faults in all other parts of the processor core. Note that reducing the voltage and frequency cause a small increase the rate of occurrence of soft errors. The detection and subsequent recovery due to soft error impose an additional energy and performance overhead. However current soft error rates are still sufficiently low enough that this additional power and performance overhead is negligible.

V. Evaluation

A. Simulation Methodology

We use a modified version of the SESC cycle accurate simulator [7]. Each program is simulated in two steps. The first step uses SESC as an execution driven simulator to provide a trace of load and branch instruction retirement events, and fingerprint generation events from the primary core. The second step uses SESC for trace based simulation of the redundant core. If during the simulation of the redundant core, any of the storage structures like the branch outcome queue or the load value queue become full, then we assume that the primary core is completely stalled for the duration for which the queue is full and delay all events generated by the primary core by the duration of the stall[2]. Instructions executing on misspeculated paths are fully simulated.

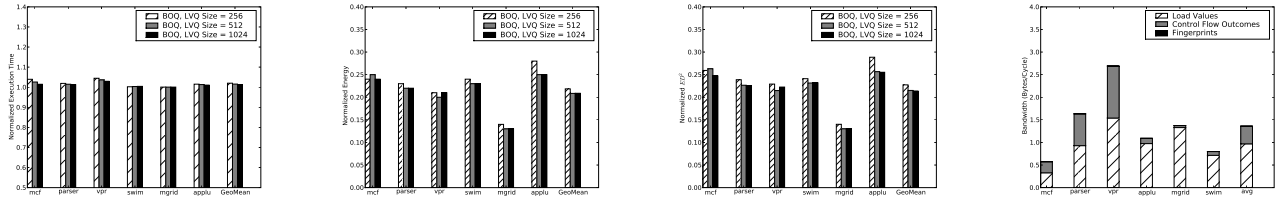
A source of inaccuracy in our trace based simulation is that it fails to account for contention for the shared bus that connects the L2 caches to memory[3].

Power estimation is carried out using a simple first order model. It is well known that dynamic power dissipation in a CMOS circuit [31] is given by:

$$P = ACV^2 f \tag{1}$$

[2]Note that this is a conservative assumption as a queue full event will only prevent the primary core from retiring instructions; execution of instructions can still continue. Our simulations show that the average duration of stalls due to queue full events is only a few tens of cycles, making it quite likely that the latency of many of these stalls can be completely hidden.

[3] All traffic between L2 and memory for the R-core is due to I-cache misses.



(a) Normalized execution time vs Queue sizes

(b) Normalized energy vs Queue sizes

(c) Normalized ED^2 vs Queue sizes

(d) Interconnect Bandwidth

Fig. 4. Performance, Energy and ED^2 for different queue sizes. In all cases: $LowThreshold = 16$, and $HighThreshold = Size/8$. Interconnect bandwidth requirements are shown for the case when queue sizes are all set to 1024 elements.

Here A is the switching factor, C is the effective switching capacitance, V is the supply voltage and f is operating frequency. A common assumption is that voltage scales linearly with frequency [5] [9] [29].

If the execution is divided into N intervals $1, 2, 3 \ldots i \ldots N$, each of length ΔT_i, and interval i operates at frequency $k_i f$ and voltage $k_i V$, where k_i is the scaling factor for interval i, then the ratio of E/E', where E is the energy consumed after frequency scaling and E' is the energy consumed without frequency scaling, is given by:

$$E'/E = \frac{\sum_{i=1}^{N} k_i^3 \Delta T_i}{\sum_{i=1}^{N} \Delta T_i} \quad (2)$$

Our simulation tool estimates the ratio E'/E using the above equation. Results in [5] show that the above is a fairly accurate model for dynamic power estimation. The assumption made by this simple model is that power consumption in the LVQ and BOQ is equal to the power consumption in the data cache and branch predictor structures. This is a conservative and reasonable assumption because the size of the LVQ and BOQ is less than the size of the data cache and branch predictor, which are turned-off and not accessed.

B. Workload

We use three integer and three floating point benchmarks from the SPEC CPU 2000 benchmark suite. To reduce simulation times, the integer benchmarks `mcf, parser` and `vpr` are simulated using MinneSPEC [10] reduced input sets. The floating point benchmarks `applu, mgrid` and `swim` are simulated using the early SimPoints given in [19].

C. Results

Effect of Queue Sizes on Performance: Figure 4(a) shows the ratio of execution time with redundant execution to execution when no redundant execution is performed (normalized execution time) for different BOQ and LVQ sizes. The mean performance degradation across all workloads for a 256, 512 and 1024 entry sized queues are 2.0%, 1.6% and 1.2% respectively. It can be seen that the integer workloads suffer greater performance degradation as compared to the floating point workloads. The least performance overhead of less than 0.1% is seen in `swim`. The highest performance overhead is 4.5% for `vpr`.

As expected, for most benchmarks the performance overhead decreases when the sizes of the queues are increased. However for the benchmark `swim`, there is an decrease in performance for larger queue sizes - queue sizes of 256, 512 and 1024 have performance overheads of 0.2%, 0.3% and 0.4% respectively. It will be shown in the next subsection that increasing the threshold values tends to increase the performance overhead. For the results in shown in Figure 4(a), as the queue sizes are increased the high threshold is also proportionally increased. Our simulation with queue sizes of 512 and 1024 with the high threshold set to 32 had a performance overhead of 0.2% only; showing that for `swim` the increase in performance due to larger queue sizes is negated by the decrease in performance due to higher thresholds.

Effect of Queue sizes on Energy: Figure 4(b) shows the normalized energy metric defined in equation (2). The mean normalized energy for BOQ and LVQ sizes of 256, 512 and 1024 are 0.22, 0.21 and 0.21 respectively, showing that our scheme achieves an mean energy reduction of almost 80%.

The highest savings are achieved in `mgrid`, which shows a decrease in energy of about 85%, while the lowest savings are seen in `applu`, which shows energy savings between 72-75%.

Effect of Queue Sizes on ED^2: Figure 4(c) shows the variation of the $Energy \times Delay^2$ metric for different BOQ and LVQ sizes.

These results are similar to the results for energy because the normalized execution times (delay) are very close to 1. We observe that the mean reduction in ED^2 is 77%, 78%, and 79% for BOQ and LVQ sizes of 256, 512 and 1024 respectively.

TABLE I
PROCESSOR AND MEMORY SYSTEM MODELED

Fetch/Issue/Retire width	6/3/3
ROB size	128
I-window	64
LD/ST queue	48
Mem/Int/FP Units	2/3/2
Branch Predictor	Hybrid
BTB	2k, 2-way
I-cache	32k/64B/4-way/2 cycles
D-cache	64k/64B/4-way/2 cycles
L2	512k/64B/16-way/14 cycles Unified, Private
Memory	450 cycles
V_{DD}	0.5-1V, steps of 0.125V
Frequency	max 3 GHz; scaled linearly with V_{DD}
Interconnect Latency	16 cycles
Queue size sampling interval (T_s)	8.33 μs (25,000 cycles at 3 GHz)
V_{DD} change latency	0.125V / 100 ns

Effect of Different High Thresholds on Performance: Table II shows the effect of increasing the high thresholds on performance. The numbers presented here are the mean normalized execution times across all benchmarks. We see that a higher threshold actually has an adverse effect on performance. The reason is that when the P-

core switches from a phase of low IPC to one of higher IPC, the queue sizes start building up. By setting the thresholds lower, we react faster to a potential change of program phase, and so this has a lower overhead on performance.

Queue Size	$Size/8$	$Size/4$	$Size/2$
256	1.020	1.023	1.026
512	1.016	1.017	1.020
1024	1.012	1.013	1.019

TABLE II
NORMALIZED EXECUTION TIME VS. QUEUE SIZES. THE THREE COLUMNS SHOW NORMALIZED EXECUTION FOR DIFFERENT THRESHOLD VALUES.

We found that changing the low thresholds did not have any significant effect on performance.

Bandwidth Requirements: Figure 4(d) shows the bandwidth requirements of our scheme. The average bandwidth requirement is about 1.4 bytes per cycle for the benchmarks we study.

VI. CONCLUSION AND FUTURE WORK

This paper has presented an initial investigation of a power-efficient microarchitecture for redundant execution on CMPs. Our results indicate that this architecture combined with our simple DVFS algorithm can achieve energy savings of up to 85%, mean savings being 79% with a mean performance overhead of only 1.2%.

In future work we intend to explore more sophisticated algorithms for DVFS, examine the effects of using DVFS on the P-core and build an execution driven simulator for our approach to obtain more accurate performance and power estimates.

VII. ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their comments and suggestions which helped improve the quality of this paper. This work is partially supported by the Swedish Foundation for International Cooperation in Research and Higher Education.

REFERENCES

[1] Todd Austin. DIVA: A Reliable Substrate For Deep Submicron Microarchitecture Design. *Proceedings of the 32nd MICRO, Nov 1999*.

[2] Dan Ernst, Nam Sung Kim, Shidhartha Das, Sanjay Pant, Rajeev Rao, Toan Pham, Conrad Ziesler, David Blaauw, Todd Austin, Krisztian Flautner, and Trevor Mudge. Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation. In *Proceedings of the 36th MICRO, Dec. 2003*.

[3] Stijn Eyerman, Lieven Eeckhout, Tejas Karkhanis, and James E. Smith. A Performance Counter Architecture for Computing Accurate CPI Components. *Proceedings of the 12th International Conference on Architectural support for Programming Languages and Operating Systems, Oct. 2006*.

[4] Mohamed Gomma, Chad Scarbrough, T. N. Vijaykumar, and Irith Pomeranz. Transient-Fault Recovery for Chip Multiprocessors. *Proceedings of the 30th ISCA, June 2003*.

[5] Canturk Isci, Alper Buyuktosunoglu, Chen-Yong Cher, Pradip Bose, and Margaret Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. *Proceedings of the 39th MICRO, Dec. 2006*.

[6] J. Dorsey et al. An Integrated Quad-core Opteron processor. *International Solid State Circuits Conference, 2007*.

[7] J. Renau et al. SESC Simulator. http://sesc.sourceforge.net/, 2005.

[8] Seongwoo Kim and Arun K. Somani. SSD: An Affordable Fault Tolerant Architecture for Superscalar Processors. In *PRDC '01: Proceedings of the 2001 Pacific Rim International Symposium on Dependable Computing*.

[9] Wonyoung Kim, Meeta S. Gupta, Wei Gu-Yeon, and David Brooks. System level analysis of fast, per-core DVFS using on-chip switching regulators. *Proceedings of the 14th HPCA, Feb. 2008*.

[10] A. J. KleinOsowski and David J. Lilja. MinneSPEC: A New SPEC Benchmark Workload for Simulation-Based Computer Architecture Research. *IEEE Computer Architecture Letters, Jan. 2002*.

[11] Sumeet Kumar and A. Aggarwal. Speculative instruction validation for performance-reliability trade-off. *Proc. of the 14th HPCA, Feb. 2008*.

[12] Benjamin Lee and David Brooks. Effects of Pipeline Complexity on SMT/CMP Power-Performance Efficiency. *Workshop on Complexity Effective Design in conjunction with 32nd ISCA, June 2005*.

[13] Yingmin Li, David Brooks, Zhigang Hu, and Kevin Skadron. Performance, Energy, and Thermal Considerations for SMT and CMP Architectures. *Proceedings of the 11th International Symposium on High-Performance Computer Architecture Feb 2005*.

[14] H. M. Mathis, A. E. Mericas, J. D. McCalpin, R. J. Eickemeyer, and S. R. Kunkel. Characterization of simultaneous multithreading (SMT) efficiency in POWER5. *IBM Journal of Research and Development, July/Sept 2005*.

[15] Pablo Montesinos, Wei Liu, and Josep Torrellas. Using Register Lifetime Predictions to Protect Register Files against Soft Errors. *IEEE Transactions on Dependable and Secure Computing, 2008*.

[16] S. S. Mukherjee, M. Kontz, and S. K. Reinhardt. Detailed Design and Evaluation of Redundant Multithreading Alternatives. *Proceedings of the 29th ISCA, May 2002*.

[17] Kunle Olukotun, Basem Nayfeh, Lance Hammond, Ken Wilson, and Kunyung Chang. The Case for a Single-Chip Multiprocessor. *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems, Oct. 1996*.

[18] Ishwar Parulkar, Alan Wood, James C. Hoe, Babak Falsafi, Sarita V. Adve, and Josep Torrellas. OpenSPARC: An Open Platform for Hardware Reliability Experimentation. *Fourth Workshop on Silicon Errors in Logic-System Effects (SELSE), April 2008*.

[19] Erez Perelman, Greg Hamerly, and Brad Calder. Picking Statistically Valid and Early Simulation Points. *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques, September 2003*.

[20] P. Racunas, K. Constantinides, S. Manne, and S. S. Mukherjee. Perturbation-based Fault Screening. *Proceedings of the 13th HPCA, Feb. 2007*.

[21] S. K. Reinhardt and S. S. Mukherjee. Transient Fault Detection via Simultaneous Multithreading. *Proceedings of the 27th ISCA, June 2000*.

[22] Eric Rotenberg. AR-SMT: A Microarchitectural Approach to Fault Tolerance in a Microprocessor. *Proceedings of Fault-Tolerant Computing Systems (FTCS), 1999*.

[23] Ruchira Sasanka, Sarita V. Adve, Yen-Kuang Chen, and Eric Debes. The energy efficiency of CMP vs. SMT for multimedia workloads. *Proceedings of the 18th Annual International Conference on Supercomputing, 2004,*.

[24] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. Automatically Characterizing Large Scale Program Behavior. *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2002), Oct. 2002*.

[25] P. Shivakumar, M. Kistler, S. Keckler, D. Burger, and L. Alvisi. Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic. *Proceedings of the International Conference on Dependable Systems and Networks, 2002*.

[26] J. C. Smolens, Brian T. Gold, Babak Falsafi, and James C. Hoe. Reunion: Complexity-Effective Multicore Redundancy. *Proceedings of the 39th MICRO, Dec 2006*, .

[27] Jared C. Smolens, Brian T. Gold, Jangwoo Kim, Babak Falsafi, James C. Hoe, and Andreas G. Nowatzyk. Fingerprinting: Bounding soft error detection latency and bandwidth. *Proceedings of the 9th ASPLOS, Oct. 2004*, .

[28] Viswanathan Subramanian, Mikel Bezdek, Naga D. Avirneni, and Arun Somani. Superscalar Processor Performance Enhancement through Reliable Dynamic Clock Frequency Tuning. In *DSN '07: Proceedings of the 37th DSN*.

[29] Radu Teodorescu and Josep Torrellas. Variation-Aware Application Scheduling and Power Management for Chip Multiprocessors. *Proceedings of the 35th ISCA, June 2008*.

[30] N. Wang and S. Patel. ReStore: Symptom Based Soft Error Detection in Microprocessors. *Proceedings of the International Conference on Dependable Systems and Networks, 2005*.

[31] N. H. E. Weste and D. Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison Wesley, 2005.