Power Efficient Redundant Execution for Chip Multiprocessors

Pramod Subramanyan Virendra Singh

***Abstract*—This paper describes the design of a power efficient microarchitecture for transient fault detection in chip multiprocessors (CMPs). Our architecture utilizes the load value queue and branch outcome queue structures first introduced in the context of simultaneous and redundantly threaded (SRT) processors. We show that these structures can be combined with our algorithm for per-core dynamic voltage and frequency scaling (DVFS) to significantly reduce dynamic power dissipation in the redundant core. 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 average of 65% and a maximum of 70% with an associated performance overhead which is less than 5%.**

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 [12], [33]. One set of approaches to the soft error problem attempt to deduce the occurrence of an error by monitoring and identifying deviations or perturbations in program behavior [13], [14]. Another set of approaches employ some form of redundant execution coupled with input replication and output comparison [2], [3], [1], [5], [6], [15], [16], [7]. Architectures like Restore [13] and Perturbation Based Fault Screening [14] 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% [1], [3], [2], [24].

While RMT remains an attractive option for transient fault tolerance, the base assumption made in RMT designs is a simultaneously multithreaded processor. Our work aims to provide transient fault detection with a small performance and power penalty *without* the assumption of a simultaneously multithreaded base processor. Such an architecture might be interesting because it alleviates the need for a wide and deep superscalar pipeline that is required for obtaining optimal power-performance efficiency from an SMT processor [22]. SMT processors also require additional area and incur additional circuit delays which negatively affect the clock rate [20], [21]. A CMP with private caches would also be less affected by destructive interference among different threads, which in some cases can negate the benefits of SMT [23].

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 because the R-core never misspeculates and never accesses the data cache, obtaining all values from the load value queue instead. In this scenario, the R-core is made to operate at a lower frequency than the P-core reducing dynamic power dissipation by a significant amount, but with only a small amount of performance degradation.

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

II. Related Work

Fault Tolerance based on RMT: Transient fault detection using simultaneous multithreading was introduced by Rotenberg in AR-SMT [1] and Reinhardt and Mukherjee in SRT [2]. SRT augments SMT processors with the branch outcome queue and load value queue structures. 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 [3], which extends SRT to simultaneously multithreaded chip multiprocessors. Gomaa et al. studied Chip Level Redundant Threading with Recovery (CRTR) [24], 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 Elison (DDBCE).

A number of variants on SRT processors have been proposed. An example is Speculative Instruction Validation (SpecIV) [7] proposed by Kumar and Aggarwal 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. However, SpecIV obtains a reduction in the number of instructions executed at the cost of slightly reduced coverage. Moreover SpecIV fetches and decodes all instructions even if they are not re-executed in the redundant thread.

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 [22] find in their study of power-performance efficient designs that achieving optimal power-performance for SMT processors requires wider (e.g.: 8-way) and deeper pipelines. Since many microarchitectural structures scale in a non-linear fashion with increasing issue width [21], wider pipelines require larger area and also lead to lower clock rates [20]. A study by Sasanka et al. [27] found that CMPs outperform SMT architectures for multimedia workloads. Another study by Li et

al. [28] 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 between threads that sometimes causes performance degradation [23].

The above results indicate that an 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.

Fault Tolerance based on CMPs: Smolens et al. [4] tackled the problem of output comparison in CMPs. In order to reduce the amount of bandwidth required for state comparison, they introduced the technique of fingerprinting which 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. Smolens et al. also introduced Reunion [5] which provides input replication in chip multiprocessors without 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 [8].

Dynamic Voltage and Frequency Scaling: Isci et al. [11] introduced a set of policies to manage per-core voltage and power levels in. Their policies aim to maximize performance while keeping power dissipation under the power budget. Their policies are managed by either a dedicated micro-controller, or a daemon running on a dedicated core. In contrast, our design utilizes occupancy information of the additional 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] performed a detailed design of on-chip regulators showing that it is possible to perform voltage changes in time periods of the 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 [18] allows the frequency of each core to be set independently.

Reducing Energy Consumption in Fault Tolerant Architectures: Montensinos et al. [17] 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.

Components protected: Our mechanism detects errors that occur within the fetch, decode and execution units. Errors in the register file will also be detected as register updates are captured by the fingerprints. We assume that data cache, L2 cache and main memory are protected by ECC.

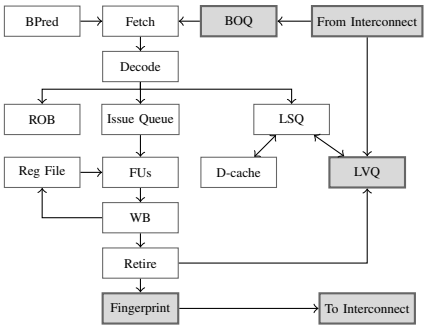

Fig. 1. Schematic diagram showing core augmented with structures required for redundant execution. Newly added structures are shaded and not used when redundant execution is not being performed.

Input replication: We need a mechanism to ensure that the two cores see exactly the same input. This is addressed by using the load value queue structure introduced by Reinhardt and Mukherjee [2]. Loads from the P-core are transferred over the interconnect and stored in the load value queue (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 on the interconnected 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 [2] the authors restrict load instructions to execute in program order, while in [3] 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 [3]. 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.[2]

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 like 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

[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.

[2]Recall that the R-core never misspeculates in the absence of a soft error.

input replication.

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* [4]. 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[3] to be different. A mismatch in fingerprints 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 [5] 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. [4] 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.

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. A possible layout for an 8-core CMP is shown in Figure 2. 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, the demands on interconnect area and power are reduced by this design.

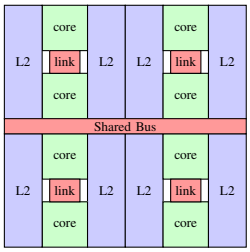

Fig. 2. Example Layout for an 8-core CMP

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 [10]. 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.

[3]For a p bit fingerprint based on cyclic redundancy codes (CRC), under the assumption that all 2^p bit values are equally likely, the probability that two different sets of updates will generate the same fingerprint is bounded by $2^{-(p-1)}$.

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[32].

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

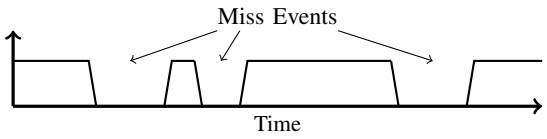

Fig. 3. Interval Analysis of Superscalar Performance. Time increases along the x-axis, and the y-axis shows the number of instructions issued.

The basis for the model is the assumption 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 3 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 4.

Let us define the lowest possible frequency at which the R-core can operate without any performance degradation as the *optimal frequency* for the R-core.

Exploiting the Effects of Time-Varying Phase Behavior: It is well known that programs have time-varying phase behavior [31]. This phase behavior causes 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

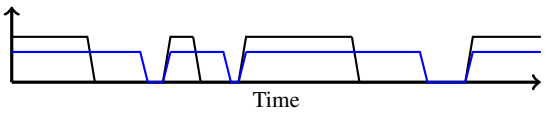

Fig. 4. Interval Analysis of the Effect of Reducing R-core Frequency. The black line shows the performance of the R-core when operating at the same frequency as the P-core. The blue line shows the performance of the R-core when operating at a reduced frequency. Reduced frequency causes the length of the intervals to increase, but since miss event resolution times are unaffected, there is no effect on performance. As in the previous figure, the x-axis is time, and y-axis is the number of instructions issued per unit time.

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 a algorithm which increases the frequency of the R-core when the queue size increases beyond a threshold, and decreases frequency when queue sizes fall below a threshold might be able to track IPC variations in P-core, reducing power dissipation without adversely affecting performance.

DVFS Algorithm: Our algorithm 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 theshold, 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. More complex and sophisticated algorithms are possible; we leave the exploration of these for future work.

IV. Evaluation

A. Simulation Methodology

We use a modified version of the SESC cycle accurate simulator [19]. 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[4].

TABLE I
Processor and Memory System Modeled

Frequency	3 GHz
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
Interconnect Latency	16 cycles
Queue size sampling interval (T_s)	8.33 μs (25,000 cycles)

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[5].

Power estimation is carried out using a simple first order model. It is well known that dynamic power dissipation in a CMOS circuit [25] is given by:

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

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 power [11] [9] [10].

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 energy is given by:

$$E' = \sum_{i=1}^{N} k_i^3 P \Delta T_i \tag{2}$$

If no voltage-frequency scaling were performed, then the energy E would have been:

$$E = \sum_{i=1}^{N} P \Delta T_i \tag{3}$$

An estimate of the reduction in energy is given by the ratio E'/E.

$$E'/E = \frac{\sum_{i=1}^{N} k_i^3 \Delta T_i}{\sum_{i=1}^{N} \Delta T_i} \tag{4}$$

Our simulation tool estimates the ratio E'/E using the above equation. Results in [11] show that the above is a fairly accurate model for dynamic power estimation.

We are in the process of developing a detailed execution driven simulation tool, but we believe our current methodology should provide a reasonable estimate of the potential benefits of our proposed architecture for this initial investigation.

[4]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.

[5]All traffic between L2 and memory for the redundant core is due to instruction cache misses; our measurements showed that traffic due to instruction cache misses was less than 10% of total traffic in all our benchmarks.

Table I lists the parameters of the simulated CMP.

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 [29] reduced input sets. The floating point benchmarks `applu`, `mgrid` and `swim` are simulated using the early SimPoints given in [30].

C. Results

Effect of BOQ and LVQ Sizes on Performance: Figure 5 shows the performance degradation 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 is seen in the benchmark of swim of less than 0.1%. The highest performance overhead is 4.5% for vpr.

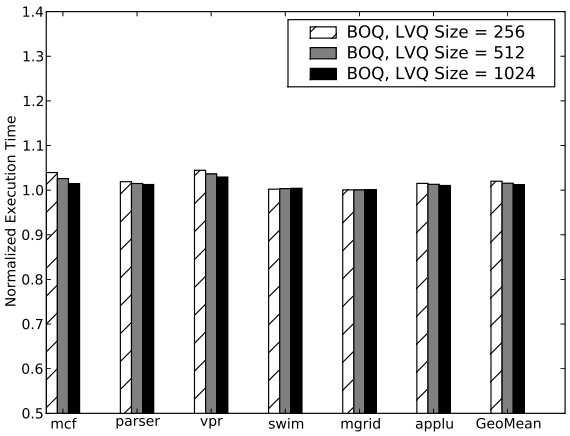

Fig. 5. Normalized execution time vs BOQ and LVQ sizes for different applications The low thresholds for this experiment are all set to 16, while the high thresholds are all set to $Size/8$.

Effect of BOQ and LVQ Sizes on Energy: Figure 6 shows the normalized energy metric defined in equation (4). 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 BOQ and LVQ Sizes on ED^2: Figure shows the variation of the $Energy \times Delay^2$ metric defined in [26] with 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.

V. Conclusions

TODO.

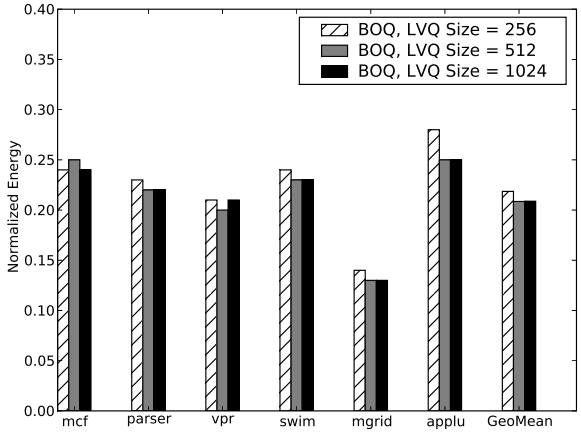

Fig. 6. Normalized energy vs BOQ,LVQ sizes. Low thresholds = 16, High Thresholds = Size/8.

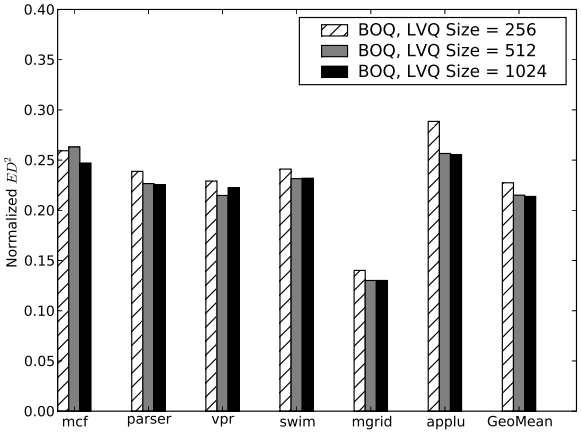

Fig. 7. Normalized ED^2 vs BOQ, LVQ sizes for different benchmarks. Low thresholds = 16, High Thresholds = Size/8.

References

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

[2] S. K. Reinhardt., S. S. Mukherjee, *Transient Fault Detection via Simultaneous Multithreading.* Proceedings of the 27th International Symposium on Computer Architecture, June 2000.

[3] S. S. Mukherjee, M. Kontz and S. K. Reinhardt, *Detailed Design and Evaluation of Redundant Multithreading Alternatives*. Proceedings of the 29th International Symposium of Computer Architecture, May 2002.

[4] J. C. Smolens et al. *Fingerprinting: Bounding soft error detection latency and bandwidth*. Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems, Oct 2004.

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

[6] Nidhi Aggarwal, P. Ranganathan, Norman P. Jouppi and James E. Smith *Configurable isolation: Building high availability systems with commodity multi-core processors.* Proceedings of the 34th International Symposium on Computer Architecture, June 2007.

[7] Sumeet Kumar, A. Aggarwal, *Speculative instruction validation for*

performance-reliability trade-off. Proceedings of the 14th International Symposium on High Performance Computer Architecture, Feb 2008.

[8] Nidhi Aggaral, J. E. Smith, K. K. Saluja, Normal Jouppi and P. Ranganathan, *Implementing High Availability Memory with a Duplication Cache*. Proceedings of the 41st International Symposium on Microarchitecture, Nov 2008.

[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 International Symposium on High Performance Computer Architecture, Feb 2008.

[10] Radu Teodorescu and Josep Torrellas, *Variation-Aware Application Scheduling and Power Management for Chip Multiprocessors*. Proceedings of the 35th International Symposium on Computer Architecture, June 2008.

[11] 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 Annual International Symposium on Microarchitecture Dec 2006.

[12] 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.

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

[14] P. Racunas, K. Constantinides, S. Manne and S. S. Mukherjee, *Perturbation-based Fault Screening*. Proceedings of the 13th International Conference on High Performance Computer Architecture, Feb 2007.

[15] Todd Austin, *DIVA: a reliable substrate for deep submicron microarchitecture design*. Proceedings of the 32nd International Symposium on Microarchitecture, Nov 1999.

[16] M. A. Gomaa and T. N. Vijaykumar, *Opportunistic transient-fault detection*. Proceedings of the 32nd International Symposium on Computer Architecture, June 2005.

[17] 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.

[18] J. Dorsey et al, *An integrated quad-core Opteron processor*. International Solid State Circuits Conference

[19] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, K. Strauss, S. Sarangi, P. Sack and P. Montesinos, "SESC Simulator", January 2005. http://sesc.sourceforge.net/.

[20] James Burns and Jean-Luc Gaudiot, *Area and System Clock Effects on SMT/CMP Processors*. Proceedings of the International Conference on Parallel Architectures and Compilation Techniques, 2001.

[21] 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.

[22] 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.

[23] 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.

[24] Mohamed Gomma, Chad Scarbrough, T. N. Vijaykumar and Irith Pomeranz, *Transient-Fault Recovery for Chip Multiprocessors*. Proceedings of the 30th International Symposium on Computer Architecture, June 2003.

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

[26] David M. Brooks, Pradip Bose, Stanley E. Schuster, Hans Jacobson, Prabhakar N. Kudva, Alper Buyuktosunoglu, John-David Wellman, Victor Zyuban, Manish Gupta and Peter W. Cook, *Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors*. IEEE Micro, Dec 2000.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] Stijn Eyerman, Lieven Eeckhout, Tejas Karkhanis, 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.

[33] Shubhendu S. Mukherjee, Joel Emer and Steven Reinhardt, *The Soft Error Problem: An Architectural Perspective*. Proceedings of the 11th International Symposium on High-Performance Computer Architecture, Feb 2005.