Appears in the Proceedings of the 40th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2010)

Energy-Efficient Fault Tolerance in Chip Multiprocessors Using Critical Value Forwarding

Pramod Subramanyan Virendra Singh
Indian Institute of Science
Bangalore, India
{*pramod@rishi.serc, viren@serc*}*.iisc.ernet.in*

Kewal K. Saluja
University of Wisconsin-Madison
Madison, WI
saluja@engr.wisc.edu

Erik Larsson
Linköping University
Linköping, Sweden
erila@ida.liu.se

Abstract

Relentless CMOS scaling coupled with lower design tolerances is making ICs increasingly susceptible to wear-out related permanent faults and transient faults, necessitating on-chip fault tolerance in future chip microprocessors (CMPs). In this paper we introduce a new energy-efficient fault-tolerant CMP architecture known as Redundant Execution using Critical Value Forwarding (RECVF). RECVF is based on two observations: (i) forwarding critical instruction results from the leading to the trailing core enables the latter to execute faster, and (ii) this speedup can be exploited to reduce energy consumption by operating the trailing core at a lower voltage-frequency level. Our evaluation shows that RECVF consumes 37% less energy than conventional dual modular redundant (DMR) execution of a program. It consumes only 1.26 times the energy of a non-fault-tolerant baseline and has a performance overhead of just 1.2%.

1. Introduction

Over the last three decades, continued scaling of silicon fabrication technology has permitted exponential increases in the transistor budgets of microprocessors. In the past, higher transistor counts were used to increase the performance of single processor cores. However increasing complexity and power dissipation of these cores forced architects to turn to chip multiprocessors (CMPs) in order to deliver increased performance at a manageable level of power and complexity. While deep sub-micron technology is enabling the placement of billions of transistors on a single chip, it also poses unique challenges. ICs are now increasingly susceptible to soft errors [25], wear-out related permanent faults and process variations [2, 5].

Traditionally, high availability systems have been restricted to the domain of mainframe computers or specially designed fault-tolerant systems [4, 14]. However, the trend towards unreliable components means that fault tolerance is now important for the commodity market as well [1]. Fault-tolerant solutions for the commodity market have different requirements and present a different set of design challenges. The commodity market requires *configurable* [1] and *low cost* fault tolerance. CMPs are appealing in this context as they inherently provide replicated hardware resources which can be exploited for error detection and recovery. A number of proposals [1, 10, 11, 17, 19, 20, 26–31] have attempted to take advantage of these aspects of CMPs to provide fault tolerance.

An important aspect of fault-tolerant CMP designs is their energy-efficiency. Power and peak temperature are key performance limiters for modern processors [12]. Since the power budget for a chip is fixed, decreasing the power consumed in any core increases the power available to other cores. This enables them to operate at a higher frequency, increasing overall system performance. Furthermore, reducing power dissipation has an additional advantage of reducing operating temperatures, which can increase chip lifetimes by an order of magnitude [7]. Reducing the energy overhead of fault tolerance schemes is also important from the point of view of data center energy. Data center energy consumption is expected to reach an unprecedented level of 100 billion kilowatt hours by 2011. Unreliable chip components would imply that a significant fraction of future data centers would require fault tolerance mechanisms to cope with hardware faults. Clearly, there is a pressing need for energy-efficient fault-tolerant architectures for future microprocessors.

In this paper, we propose Redundant Execution using Critical Value Forwarding (RECVF), an architecture for energy-efficient fault-tolerant CMPs. RECVF executes one logical thread on two cores of a CMP. One of these cores is designated as the leading core, while the other is designated as the trailing core. The first contribution of this paper is the introduction of the idea of *critical value forwarding* (CVF). In an RECVF processor, the leading core *assists the execution* of the trailing core by forwarding the results of instructions on the *critical path*. CVF breaks data dependence chains in the trailing core because the results of instructions on the critical path are made available to the trailing core even before they complete execution. This in turn allows instructions dependent on these instructions to execute earlier, creating a cascade effect that improves the performance of the trailing core.

RECVF solves the following key challenges in design of such an architecture:

1) Identifying instructions on the critical path. The challenge here is to identify a few critical instructions that have the most impact on performance.
2) Designing mechanisms for transferring the results of these instructions from the leading to the trailing core.
3) Validating the forwarded values in the trailing core to

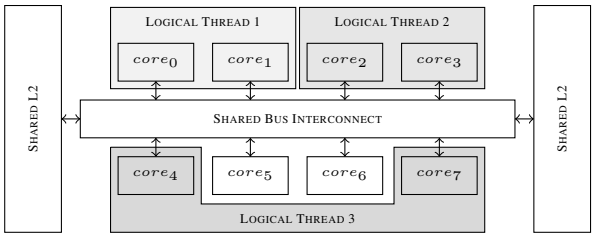


Figure 1 – CMP Block Diagram

ensure correct operation even in the presence of an error in the forwarded values.

Our second main contribution is to combine the idea of critical value forwarding with that of per-core Dynamic Voltage-Frequency Scaling (DVFS) [12, 13]. This allows the trailing core to execute at a much lower frequency than the leading core, significantly reducing the energy overhead of redundant execution. We propose two new algorithms for per-core DVFS in this context and examine the energy savings due to these.

We evaluate RECVF extensively and compare it with two previous proposals for energy-efficient fault-tolerant CMPs. Our evaluation shows that for a conventional CMP with a shared L2 cache, RECVF has a performance loss of less than 1.2% and consumes 1.26 times the energy of the non-fault-tolerant baseline processor. In comparison, the Parallelized Verification Architecture (PVA) proposed by Rashid et al. [20] has a performance loss of 4.7% and consumes 1.32 times the energy of the baseline. Mukherjee and Reinhardt's Chip-level Redundantly Threaded (CRT) [19] processor has a performance loss of 4.6% and an energy consumption of 1.52 times the baseline. For a future CMP architecture with private L2 caches and higher interconnect latencies, RECVF has a performance loss of 3.9% and an energy consumption of 1.45 times the baseline processor. In comparison, PVA and CRT have a performance loss for 10.4% and 9.3%, and consume 1.62 and 1.92 times the energy of the baseline processor respectively.

2. Description of Architecture

RECVF provides fault tolerance by executing a single logical thread on two cores of CMP. One of these cores is designated as the leading core while the other is designated as the trailing core. The two cores are assumed to be connected by a shared interconnect. The leading and trailing cores exchange information over this interconnect. In our implementation, we use a shared bus as the interconnect. However, RECVF is amenable to implementation over more complex interconnects such as NoCs.

Figure 1 shows an eight core RECVF CMP executing three logical threads requiring fault tolerance. The configuration shown depicts each core with a private L1 cache and all the cores sharing a single L2 cache. RECVF can be used in implementations with private L2 caches as well. The following subsections describe the operation of each component of an RECVF processor.

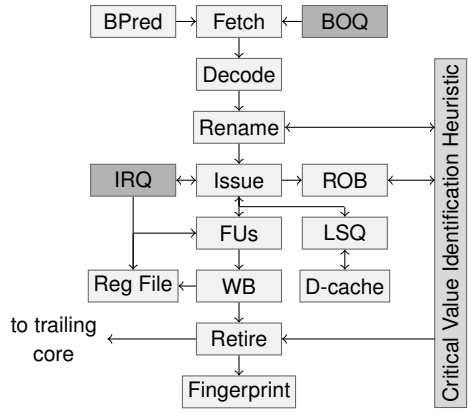


Figure 2 – An RECVF processor core.

2.1. Core Architecture

Figure 2 shows the block diagram of an RECVF processor core. The processor pipeline is augmented with three additional structures, the branch outcome queue (BOQ), the instruction result queue (IRQ) and circuitry implementing a critical value identification heuristic. The BOQ and IRQ are used only in the trailing core, while critical value identification is performed only in the leading core.

2.1.1. Identifying Critical Values. Our approach to identifying critical path instructions is similar to that proposed in [32]. The basic idea is to mark an instruction as critical if it satisfies certain *marking criteria* during its execution. We evaluated a number of critical value identification heuristics based on this principle. A list of these is shown in table 1.

2.1.2. Handling Branch/Jump Instructions. Branch/Jump instructions are handled differently for the purposes of critical value identification. RECVF marks mispredicted branch instructions as critical. The target addresses of mispredicted branches are forwarded from the leading to the trailing core. In the trailing core, these addresses are used as predictions. As will be seen in §2.5, this mechanism provides almost the same speedup as forwarding the results of all branch instructions, but requires very little bandwidth.

2.2. Operation of the Leading Core

With the exception of critical value marking and forwarding, the leading core operates like conventional superscalar processor cores. Critical value forwarding is done after instruction retirement.

Both leading and trailing core execute instructions in chunks. When the leading core finishes the execution of a chunk it requests the trailing core to execute that chunk. An instruction index within the current chunk is forwarded along with the value by the leading core. This index is used by the trailing core to map forwarded instruction results to instructions.

	Heuristic	Marking Criteria	Rationale
1	$robStall$	Instruction at the head of ROB prevents retirement because it is not yet executed.	Instructions that are unable to execute until they reach the head of the ROB are likely to be on the critical path.
2	$instQHead$	Instruction reaches head of instruction queue before being selected for execution.	Instructions unable to execute until they reach the head of of the instruction queue are likely to be on the critical path.
3	$instQHFree$	Instruction produces a value that frees an instruction at the head of its queue.	Forwarding this value may help the dependent instructions execute earlier in the trailing core.
4	$freedN$	Instruction freed at least N instructions for execution when it completed.	An instruction that frees a large number of other instructions for execution is more likely to be on the critical path.
5	$fanoutN$	Instruction produces a value that is used by at least N other in-flight instructions.	An instruction that produces a value that a large number of other instructions use is likely to be on the critical path.
6	$everyN$	Every Nth instruction is marked as critical.	A simple heuristic that serves as a benchmark for comparison against more sophisticated heuristics.
7	$allBJ$	All branch/jump instruction outcomes are forwarded.	This policy estimates the speedup obtained by forwarding just branch instructions.
8	$mispredBJ$	Only mispredicted branch/jump instruction outcomes are forwarded.	This policy compares the loss in speedup due to forwarding mispredicted branch outcomes in comparison to forwarding all branch outcomes.
9	$loadsOnly$	Only mispredicted branches and load values are forwarded.	This is the baseline for full load replication (FLR). (See §2.4.)
10	all	All possible values are forwarded.	Corresponds to an oracle heuristic given infinite storage space and infinite bandwidth.

Table 1 – Description of Critical Instruction Identification Heuristics

2.3. Operation of the Trailing Core

2.3.1. Operation of the BOQ. The trailing core stores the branch outcomes it receives in the branch outcome queue (BOQ). Unlike previous implementations of the BOQ our implementation is different because it does not store the targets of all branch instructions. A branch outcome is mapped to a branch instruction using the index transmitted by the leading core. When a branch instruction is fetched, if the target address is present in the BOQ, then this outcome is used to override the output of the branch predictor.

2.3.2. Operation of the IRQ. The trailing core stores the results of instructions other than branch instructions in the Instruction Result Queue (IRQ). Like the BOQ, the IRQ also stores an index along with the value to map instruction results to instructions. At the time of dispatch, the IRQ is examined to see if the result of this particular instruction is available. If so, the IRQ is read and its value is written into the register file. This allows dependent instructions of this instruction to begin execution immediately.

2.4. Options for Input Replication

An important issue that needs to be addressed in a system for redundant execution is how inputs to the two cores are replicated. In any fault-tolerant CMP proposal that does not use lockstepped execution, there is a delay between the time a load instruction is executed by the leading core and the time it is executed by the trailing core. Between these two events, a different processor may modify the value stored in the memory location addressed by the load. This may cause the trailing core to read an incorrect value, resulting in a problem referred to as the *input incoherence problem*.

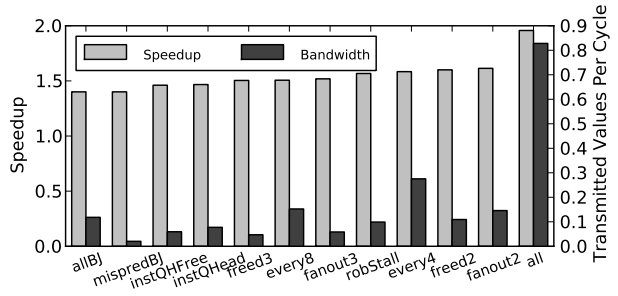
The default implementation of RECVF, which we refer to as partial load replication (PLR), fully re-executes most load instructions in the trailing core. The only load instructions that are not re-executed are those that read from cache lines obtained from cache-to-cache transfers (see §3.2). Although we do not show the results here, experiments with the SPLASH2 benchmarks revealed that PLR re-executes 93% of all load instructions. Note that for a single-threaded program, all load instructions are fully re-executed. Hence, PLR has most of the fault-coverage of a mechanism that fully re-executes loads without the corresponding complexity and performance/energy costs.

We also study the option of full load replication (FLR). FLR works like SRT/CRT [19, 21] and replicates the results of all load instructions in the leading core and transfers them to the trailing core. This option is expected to perform better at the cost of lower fault coverage and a higher bandwidth requirement.

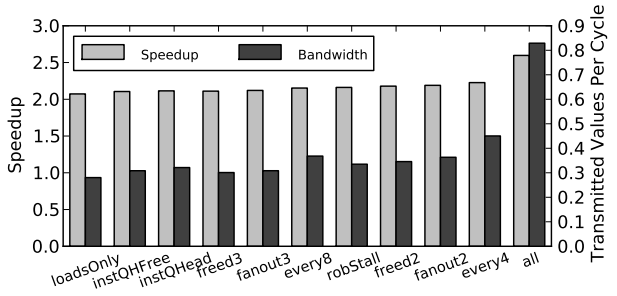
2.5. Effectiveness of Heuristics for Critical Value Identification

Figure 3 shows the performance of the critical value identification heuristics. The graph shows the mean speedup of the trailing core over the leading core averaged across the SPEC benchmark suite. Speedup is the ratio of the IPC of the trailing core to that of leading core. Both cores are operated at the nominal frequency. The IPC of the trailing core is computed only over its active period, i.e., excluding the regions of time between the completion of chunk i and the start of execution of chunk $i+1$ arrives. Note that is a conservative estimate of the speedup. Section 4.1 has further details on our methodology.

CVF has a large impact on the performance of the trailing core. The trailing core experiences speedups of up to 1.6X and 2.2X over the leading core for PLR and FLR respectively. This means that the trailing core can be operated at approximately 0.6



(a) Partial Load Replication



(b) Full Load Replication

Figure 3 – Performance of the critical value identification heuristics that we examine.

times the frequency of the trailing core for the PLR configuration, while it can be operated at less than half the frequency of the leading core for the FLR configuration. The best performing heuristic is *fanout2*, and we report results only for this heuristic.

2.6. DVFS in the Trailing Core

Critical value forwarding creates *slack* in the trailing core which can be exploited by operating the core at a lower voltage-frequency level. However, the slack is not constant for all programs. It also varies with program phases. When the leading core is executing a phase of high IPC, there is less slack to be exploited in the trailing core, and vice versa. Therefore, the challenge is to dynamically set the voltage-frequency level of the trailing core based on the program phase behavior. In this section we describe two algorithms that attempt this.

2.6.1. QSize-DVFS algorithm. This algorithm is based on the observation that the sizes of the BOQ and the IRQ are an indication of how far behind the trailing core is as compared to the leading core. Therefore, when a program goes from a phase of low IPC to one of high IPC, the trailing core will be unable to keep up, and the occupancy of these queues will increase. Such an occurrence indicates that the frequency of the trailing core ought to be increased. Conversely, if the queues are nearly empty, then it means that the trailing core is able to easily keep up with the leading core. In this scenario, the frequency of the trailing core ought to be decreased.

The *QSize-DVFS* algorithm implements exactly this idea. It maintains four thresholds: low and high thresholds for the BOQ and IRQ. Periodically, the sizes of the queues are compared to thresholds. If one of the queue sizes is less than the low threshold, then the frequency is decreased. If one of the sizes are greater than the high thresholds, then the frequency is increased.

We experimented with a number of different values for the thresholds and chose the configuration which minimized the ED^2 (energy-delay-square) product across all the benchmarks. Although we do not report the results here, we observed very little variation (less than 2%) across different threshold values.

2.6.2. IPC-DVFS algorithm. The *IPC-DVFS* algorithm takes a direct approach to determining the frequency of operation the trailing core. The idea behind this algorithm is use the ratio of the IPCs of the two cores to set the frequency of the trailing core. For example, if, for a certain period of execution, the IPC of the leading core is 1.0, while that of the trailing core is 2.0, then the trailing core ought be operated at half the frequency of the leading core.

The *IPC-DVFS* algorithm generalizes this idea in the following way. The two cores keep track of their respective IPC over the DVFS update interval. At the end of the interval, the ratio of the leading core IPC to the trailing core IPC is taken, and a scaled version of this value is used to set the frequency of the trailing core for the next interval.

3. Fault Tolerance Mechanisms

Any fault-tolerant system needs to address four important issues: fault detection, fault isolation, fault recovery and fault coverage. The following subsections discuss these topics in the context of RECVF.

3.1. Fault Detection

To detect faults, RECVF needs to compare the outputs of the leading and trailing cores that execute a single logical thread. To do this, the set of cores executing a program periodically synchronize and exchange *fingerprints* [26]. A fingerprint is a CRC-based hash of the architectural updates of the processor. The hash incorporates register updates, load store addresses and branch targets. The fingerprint is updated every cycle after instruction retirement. It is exchanged with the partner core at the time of a fingerprint comparison. Faults are detected when fingerprints are exchanged and at least one of the cores detects a mismatch in the fingerprints. If the fingerprints do not match, an error recovery operation is triggered. If the fingerprints match, then the current register state is stored in a *checkpoint store* and all lines in the cache are marked verified (See §3.2).

3.1.1. Verification of Forwarded Values. A value forwarded from the leading to the trailing core may be corrupted due to the occurrence of an error. At first glance, it appears as if we need an additional mechanism to verify the correctness of each

value that is forwarded from the leading core. However, the key observation here is that fingerprinting can be used to detect the occurrence of errors in the forwarded values. To see why, assume that an instruction i_x in the leading core forwards an erroneous value corresponding to the instruction i'_x in the trailing core. Assume without loss of generality that i_x is the earliest instruction that forwards an erroneous value to the trailing core. When i'_x executes in the trailing core, its input operands will have the correct (i.e. error-free) values and will compute the correct result. Consequently, since i_x and i'_x generate different results (one correct and one erroneous), the fingerprints computed in the two cores will be different. This enables detection of the error.

3.2. Fault Isolation

When a fault occurs in RECVF, it may be detected only when the next fingerprint comparison occurs. Between the time that the fault occurs and the time it is detected, the fault must not propagate outside the CMP or to other executing processes. This property is called *fault isolation*. In a CMP, there are two ways in which a fault can propagate outside the CMP or to other processes.

Firstly, a corrupted cache block may be replaced and written back to a lower level of the memory hierarchy, from where it can propagate to main memory or other processes. This is prevented by using a cache architecture that is similar to speculative versioning caches [16].

The cache in an RECVF CMP stores an *unverified* bit along with every cache line in the L1 data cache. Any write to a cache line sets the unverified bit. If the unverified bit is set, a cache line is deemed to be *locked* and is not allowed to be written back to a lower level of the memory hierarchy. When fingerprints are compared and found to match, the unverified bits of all lines in the cache are cleared. When a verified line is marked unverified, the line must be written back to a lower level cache. If a line needs to be replaced and all the lines in its set are locked, all the processors synchronize and a fingerprint comparison is initiated. Execution proceeds after the fingerprint comparison succeeds.

For correct execution of multithreaded workloads, the unverified bit must be transmitted along with the data when one cache supplies data to another cache. RECVF implements partial load replication (PLR) by forwarding load values from the leading to the trailing core for all cache lines obtained from cache-to-cache transfers. This requires the storage of one more bit along with each cache line in the L1 data cache. This bit is called the C2C and identifies unverified cache lines obtained from cache-to-cache transfers. It is cleared when the line is marked verified.

The second method by which a fault may propagate outside the processor is through I/O operations. RECVF forces a checkpoint to be taken and fingerprints compared before each I/O operation to ensure that I/O is done only with verified data.

3.3. Fault Recovery

When an error is detected, the two cores reset the program counter to the instruction following the last verified instruction, and restore the register state from the checkpoint store. All unverified lines in the L1 cache are invalidated and then normal execution resumes. Invalidating all unverified lines in the L1 cache ensures that any updates performed to memory are undone. Subsequent accesses to these lines will fetch the verified versions from the L2 cache.

3.4. Fault Coverage

Since, RECVF is based on spatial redundancy [21], it can detect faults that result in different architectural updates in the two cores. This includes almost all soft errors and hard errors that result in diverging architectural state across the cores. RECVF provides a high degree of coverage for processor control and execution logic. However, RECVF may not be able to cover all faults that occur in the cache coherence related circuitry because it does not redundantly access the memory hierarchy for unverified cache lines obtained from cache-to-cache transfers.

4. Evaluation

4.1. Simulation Methodology

Table 2 – CMP configuration

# of cores	8
Technology node	32 nm
Nominal frequency	3 GHz
Fetch/issue/retire	4/4/4 instructions per cycle
ROB size	128 instructions
Int/FP registers	160/128
Integer/FP window	64/32 instructions
Load/store queue	32 instructions
Mem/Int/FP units	4/6/4
I-cache	32k/64B/4-way/2 cycles
D-cache	64k/64B/4-way/2 cycles
Memory	400 cycles
Branch predictor	hybrid of bimodal/gshare 16k entries in each predictor
Branch target buffer	4k entries, 4-way set-associative
Return address stack	32 entries
BOQ size	64 entries
IRQ size	512 entries
Checkpointing interval	50,000 instructions
Checkpointing latency	64 cycles
Shared L2 configuration	
L2	16 MB/64B/8-way/40 cycles
Interconnect latency	24 cycles
Private L2 configuration	
L2	2 MB $\times$ 8/64B/8-way/24 cycles
Interconnect latency	40 cycles
Configuration for the PVA architecture of Rashid et al. [20]	
PCB size	1024 entries
PCB sections	8
PCB access hash table	257 entries
PCB access latency	8 cycles
Hash table access latency	1 cycle
LVQ size	512 entries (for CRT [19] only)

Our evaluation uses an appropriately modified version of the SESC execution-driven simulator [8]. The simulator models an out-of-order superscalar processor in a detailed manner and fully

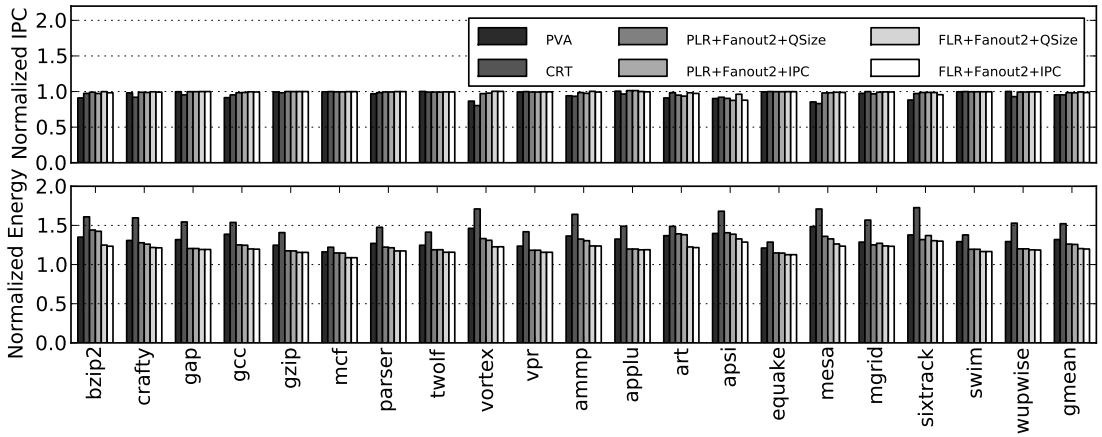


Figure 4 – Normalized IPC (above) and normalized energy (below) of the shared L2 configuration

simulates "wrong-path" instructions. Details of the CMP model are given in Table 2.

In order to put our results in context, we compare our architecture against two previous proposals: (1) the Parallelized Verification Architecture (PVA) from [20] and (2) Chip-level Redundantly Threaded (CRT) processors from [19].

To estimate energy consumption, we modified SESC's power model, which is based on Wattch [6]. We included power models for the IRQ, BOQ, the Post Commit Buffer (PCB) used in PVA and the Load Value Queue (LVQ) used in CRT. CACTI 5.3 [23] was used to model the energy of the shared L2 cache. The voltage-frequency levels for per-core DVFS used in our study are shown in Table 3. We assume fine-grained, low-latency per-core DVFS similar to the one proposed in [13]. The impact of higher-latency coarse-grained DVFS is investigated in §4.4.

Table 3 – Voltage-frequency levels for per-core DVFS

Voltage (V)	1.0	0.9	0.8	0.7	0.6	0.6
Frequency (GHz)	3.0	2.7	2.4	2.1	1.8	1.5

DVFS level change latency	$100ns$
DVFS update interval	$1\mu s$

We show results for two CMP configurations. One is a conventional CMP architecture with a 16 MB shared L2 cache. The second configuration has a 2 MB private L2 cache associated with each core. In the private-L2 configuration prefetch hints are forwarded from the leading to the trailing core caches.

Workload: We simulated ten integer and ten floating point benchmarks from the SPEC CPU 2000 benchmark suite. For each benchmark, we executed a single SimPoint [24] of length one billion instructions.

4.2. Shared L2 Configuration Results

4.2.1. IPC Results. The top half of figure 4 shows the IPC of each of the benchmarks normalized by the IPC of the baseline processor. The mean IPC degradation of PVA is 4.7%. PVA loses performance mainly due to high PCB occupancy. A full PCB can stall retirement in the leading core. CRT's mean IPC degradation is 4%. In CRT, a store cannot retire from the leading core's store buffer until it is verified by the trailing core. This creates additional pressure on the store buffer. CRT has a performance problem with `mesa` and `vortex`, both of which have a high fraction of store instructions.

The RECVF configurations exhibit mean IPC degradation varying between 0.5% and 1.4%. The worst performing benchmark for RECVF is `apsi`. `Apsi` is slowed down because of high occupancy of the IRQ for certain phases of the program. When the IRQ is full, the leading core is unable to make progress because the trailing core cannot accept the critical values sent by the leading core.

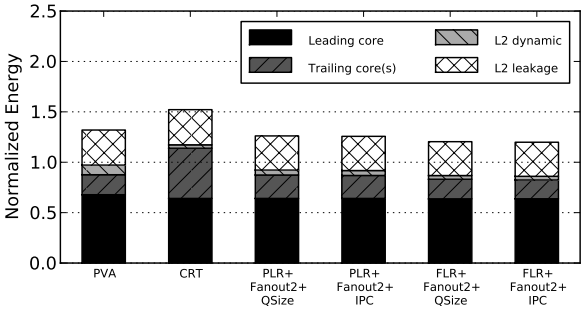


Figure 5 – Component-wise energy consumption breakdown for the architectures evaluated across all the benchmarks.

4.2.2. Energy Results. The bottom half of figure 4 shows energy consumption normalized by the energy consumption of the baseline processor. PVA consumes 1.32 times the energy of the baseline, while CRT consumes 1.52 times the energy of the baseline. The RECVF configurations using PLR consume 1.26 times the energy of the baseline. FLR is somewhat successful in trading-off lower fault coverage for reduced energy consumption with a mean energy consumption of 1.20 times the baseline.

To understand these results better, we present a breakup of the energy consumption in Figure 5.

We can make the following observations from the figure. Firstly, as one would expect, the leading cores of each configuration dissipate roughly the same amount of energy. Secondly, the trailing core in CRT consumes more energy than the trailing cores in RECVF and PVA, because the latter are running at lower voltage-frequency levels. A third interesting observation is that PVA consumes significantly higher energy in the L2 cache. This is because PVA stores verified lines in the L2 cache, which is effectively equivalent to using a write-through policy for the L1 cache.

4.3. Private L2 Configuration Results

4.3.1. IPC Results. The top half of figure 6 shows the normalized IPC for the private-L2 configuration. The mean IPC degradation for PVA with a single-ported PCB is 10.4%. `Vortex` and `sixtrack` are the worst-affected benchmarks, with IPC degradations of 22% and 19% respectively. This is due to increased PCB occupancy caused by higher interconnect latencies. CRT also performs poorly for this configuration, exhibiting a mean IPC degradation of 9.3%. This is because of increased occupancy of the store buffer. On an average, store buffer occupancy for CRT is 2.2 times the store buffer occupancy for the baseline processor. Compared to the results for the shared L2 configuration in §4.2, the problem here is exacerbated by higher interconnect latencies. *The architectures based on RECVF have a much lower mean IPC degradation for this configuration, varying between 2.2% and 3.9%.*

4.3.2. Energy Results. The bottom half of figure 6 shows the energy dissipation for the private L2 configuration. It is apparent that PVA and CRT dissipate much more energy than RECVF. PVA with a single-ported PCB consumes 1.62 times the energy of the baseline processor. CRT dissipates 1.92 times the energy of the baseline. RECVF consumes 1.45 times the energy of baseline processor for the PLR configuration. The FLR configurations consume 1.39 times the energy of the baseline.

4.4. Impact of Coarse-grained DVFS

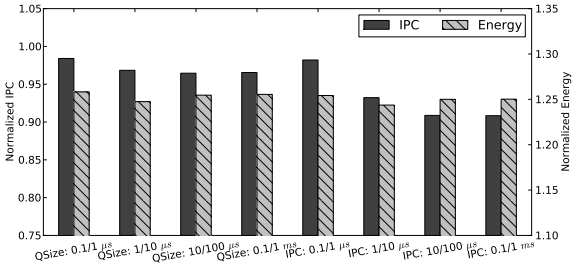


Figure 7 – Impact of the higher-latency and coarse-grained DVFS.

Our baseline architecture uses fast fine-grained DVFS similar to the proposal in [13] re-evaluating trailing core voltage-frequency level every 1 μs. In this section we evaluate the impact of more conservative per-core DVFS implementations.

Figure 7 shows the normalized IPC and energy values averaged across all the benchmarks for a number of DVFS configurations. These results are for the shared L2 configuration. The first and fourth bars in the figure correspond to our baseline DVFS architectures. The other bars show coarse-grained DVFS with parameters indicated as latency/update interval below the bars. For example, "0.1 ms/1 ms" means switching between voltage-frequency levels takes 0.1 ms, and voltage-frequency levels of the trailing core are re-evaluated every 1 ms.

IPC decreases with increasing DVFS update intervals. This is because longer update intervals find it harder to track fine-grained changes in program phase. However, energy consumption remains roughly constant across all the DVFS configurations.

The IPC-DVFS algorithm suffers a performance loss of 9% at a 1 ms update interval. In contrast, for the QSize-DVFS algorithm even with a very conservative DVFS architecture that updates the voltage-frequency level every 1 ms, the mean IPC degradation is only 4% and the increase in energy dissipation is restricted to a few percent. This is an important result, showing that RECVF is applicable even for much more conservative per-core DVFS implementations.

4.5. Impact of Limited Voltage Scaling

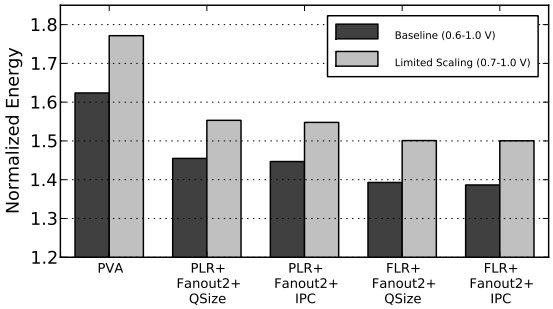


Figure 8 – Impact of limited voltage scaling

Both our proposal and PVA rely on voltage and frequency scaling in order to reduce energy dissipation. Aggressive voltage scaling might prove to be difficult in future technology nodes. Figure 8 shows the impact of reduced voltage scaling for the private L2 configuration.

PVA's energy dissipation increases by 9.1%. RECVF's PLR+QSize configuration's energy dissipation increases by only 6.8%. None of the RECVF configurations show an increase of more than 8.25%.

4.6. Bandwidth Requirements

Figure 9 shows the average core-to-core bandwidth required by each scheme in units of values per cycle. For PVA this includes the bandwidth consumed by verifying stores, PCB lookups and invalidation messages, but does not include the

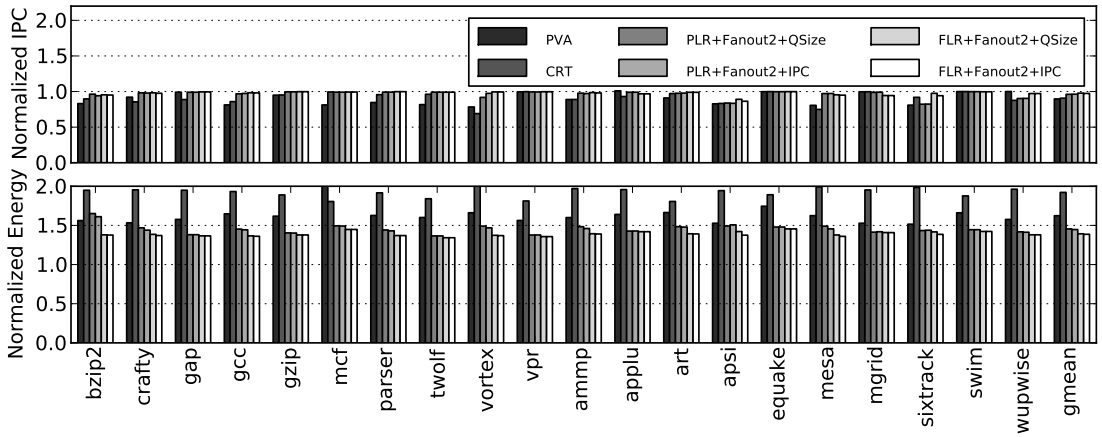


Figure 6 – Normalized IPC (above) and normalized energy (below) of the private L2 configuration

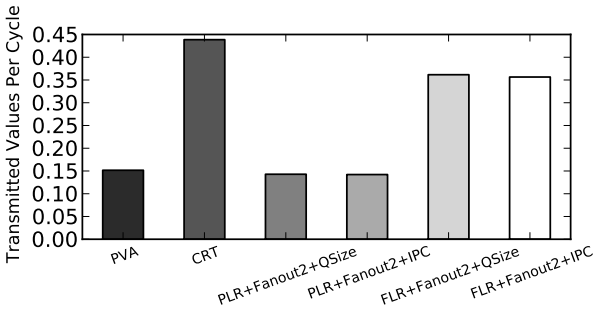


Figure 9 – Core-to-core communication bandwidth.

bandwidth required for the additional writes performed to the L2 cache.

The PLR based design for RECVF has the lowest bandwidth requirement, while CRT and FLR have the two highest requirements. PVA has a bandwidth requirement that is slightly higher than that of RECVF.

5. Related Work

Execution Assistance: Fault detection using redundant execution is a well studied topic. In particular, our work builds upon a rich body of earlier work in the area of leader/follower architectures. In such architectures, a single logical thread is executed using two physical threads, a leading and a trailing thread. Typically the leading thread assists the trailing thread in order to improve the trailing thread's performance. This idea is referred to as *execution assistance* and was introduced by Rotenberg in AR-SMT [22]. AR-SMT forwarded branch outcomes and all values as predictions to the trailing thread. Variants of this idea were also explored in DIVA [3] and SRT [21].

A large number of subsequent papers have used some form of execution assistance in the design of several interesting architectures. Table 4 presents a classification of some proposed

Values Forwarded	Proposals
All values	AR-SMT [22], DIVA [3], Slipstream[1][31], Madan et. al [17].
Loads and branches	SRT [21], CRT [19], SRTR [33], CRTR [10], SpecIV [15], EERE [29], MRE [30].
Branches only	PVA [20], Paceline [11], Decoupled performance correctness[9], Circuit pruning[18].
Critical values	RECVF

1. Slipstream's leader core forwards all values that it executes. However, it may execute a subset of program due to ineffectual instruction elision.

Table 4 – Mechanisms for Execution Assistance

mechanisms and how they compare to RECVF. The rest of this subsection discusses these design options.

While forwarding all values provides the highest possible speedup in the trailing thread, it also requires inordinately high bandwidth. As a result it is mainly suited for use within the components of a single processor core, like in the case of AR-SMT and DIVA. Forwarding all values to a different core is likely to require adjacent placement of cores. This reduces scheduling flexibility. Note that AR-SMT and Slipstream assume the presence of value-prediction support in the baseline processor to detect errors in the forwarded values. In contrast, our proposal is able to use forwarded values in the trailing core with very little additional hardware.

SRT [21] introduced the idea of forwarding only load values and branch outcomes from the leading to the trailing thread, an approach which has been adopted in a large number of subsequent proposals [10, 15, 19, 28, 30, 33]. Forwarding branch outcomes eliminates branch mispredictions while forwarding load values has two beneficial effects. Firstly, it eliminates data cache misses in the trailing thread. Secondly, it solves the problem of input incoherence. However, load and branch instructions form more than one-third of the instruction mix of the SPEC CPU 2000 benchmarks. As such forwarding the results of these instructions requires considerable bandwidth. Furthermore, all these proposals suffer from reduced fault coverage because they do not fully re-execute load instructions in the trailing core.

Forwarding only branch outcomes has also been studied in a few proposals. As shown in §2.5, CVF can provide much higher speedup of the trailing core with only marginally higher bandwidth requirements.

RECVF improves upon existing mechanisms for execution assistance by focusing on critical instructions. Our results show that nearly 80% of the speedup of forwarding all instructions can be achieved by forwarding the results of just 10-15% of all instructions.

Fault-Tolerant Architectures: Todd Austin introduced DIVA [3] which is a novel fault detection and correction architecture. DIVA 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 making it less susceptible to soft and hard errors. While the DIVA idea itself is quite robust, some implementation details are not amenable to modern deep sub-micron technologies. Firstly, DIVA cannot detect soft errors that occur in the checker processor. Secondly, resources and functional units in the checker core are unavailable for normal execution. In contrast, our architecture gainfully employs the computing resources of the trailing cores when redundant execution is disabled.

Transient fault detection using simultaneous multithreading was introduced by Rotenberg in AR-SMT [22] and Reinhardt and Mukherjee [21] in Simultaneously and Redundantly Threaded (SRT) processors. An SRT processor augments SMT processors with additional architectural structures like the branch outcome queue and load value queue for transient fault detection. Since an SRT processor provides an unpredictable combination of space and time redundancy, it cannot guarantee the detection of permanent faults. Moreover, SRT has a performance overhead of about 20-30% compared to the baseline processor, and consumes about 1.5-1.6X the energy. Muhkerjee et al. also introduced chip-level redundant threading (CRT) [19], which extends SRT to simultaneously multithreaded chip multiprocessors. Gomaa et al. studied Chip-level Redundant Threading with Recovery (CRTR) [10], which uses the state of the trailing thread to recover from an error. Our evaluation has shown that RECVF provides better performance and energy efficiency than CRT. We expect that RECVF will also outperform CRT derivatives such [10, 17] because these proposals add additional overheads to CRT to provide recovery.

Rashid et al. [20] proposed the parallelized verification architecture (PVA) for fault-tolerant CMPs which saves energy by parallelizing the verification and executing it on two cores. The verification cores are operated at half frequency and voltage levels. Our evaluation shows that that although PVA decreases trailing cores' power significantly, this comes at the expense of increased L2 cache power. RECVF has higher energy efficiency and lower performance degradation than PVA. Furthermore, PVA uses three cores to execute a single logical thread, while RECVF uses only two. Consequently, RECVF also delivers higher throughput than PVA.

Smolens et al. [26] 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 [27] 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 complex changes to the cache coherence controller which is a component that is difficult to design and verify. RECVF provides fault coverage comparable to that of Reunion at a lower complexity and energy cost.

In [28, 29], we explored the idea of per-core DVFS to reduce power in a CRT like architecture. That work was able to show power savings for programs which had poor branch predictor and L1 data cache performance by utilizing the slack created by these miss events to operate the trailing core at a lower frequency. The idea of critical value forwarding introduced in this paper does not rely on these programs properties.

In [30], we introduced the idea of Multiplexed Redundant Execution (MRE). MRE uses the idea of coarse-grained multithreading to execute multiple trailing threads on a single core, thereby improving CMP throughput. Although MRE does not directly decrease core power, it reduces overall system power by using fewer cores to redundantly execute a given set of programs.

Speculative Mechanisms for Performance Improvement: Paceline [11] is a proposal for CMP performance improvement that operates the leading core at higher than its nominal frequency. In effect, the leading core performs timing speculation, while the trailing core is used to detect errors. RECVF differs from Paceline in two important aspects. Firstly, timing speculation is orthogonal to our proposal. Secondly, critical value forwarding is more effective than branch outcome forwarding at speeding up the trailing core. Figure 3 suggests that RECVF may perform energy-efficient timing speculation more effectively than Paceline. We leave exploration of this idea for future work.

Slipstream [31] is another proposal for performance enhancement that speeds up the leading core by detecting and eliding "ineffectual instructions". The leading core speculatively removes ineffectual instructions from the execution stream, and uses the trailing core to fix up the results of misspeculation. Circuit pruning [18], and performance correctness decoupled architectures [9] are two different mechanisms that target performance improvement using leader-follower configurations. Circuit pruning prunes the hardware of the leader core to produce a smaller, but not fully correct core that can operate at a higher frequency. The pruned core assists the execution of a functionally correct trailing core. The decoupled performance correctness architecture executes a "skeleton" program on the leading core which generates branch outcomes and cache prefetches for the trailing core. Unlike RECVF, all these proposals implement some form of speculation in the leading core to improve its performance. These speculative mechanisms are orthogonal to RECVF, and can potentially be synergistically combined with RECVF to produce interesting architectures that provide energy-efficient fault-tolerance while simultaneously increasing performance.

6. Conclusion

Decreasing feature sizes, lower design tolerances and higher operating temperatures have resulted in the emergence of wear-out related permanent faults and transient faults as significant concerns in modern microprocessors.

In this paper, we showed the design of an energy-efficient fault-tolerant microarchitecture for chip multiprocessors. Our proposal introduces the idea of critical value forwarding (CVF), a technique that improves the performance of redundant execution by forwarding results of critical instructions from the leading core to the trailing core. We proposed heuristics for identifying critical values and showed how CVF can improve the performance of the trailing core without any loss in fault coverage. We proposed two algorithms for per-core dynamic voltage-frequency scaling (DVFS) to exploit the slack created by CVF.

Our evaluation showed that RECVF has a performance overhead of less than 1.2% for a shared-L2 CMP and consumed only 1.26 times the energy of the baseline processor. For a future CMP with higher latency interconnects and private L2 caches, RECVF had a performance overhead of less than 4.0% and consumed 1.45 times the energy of the baseline processor. We compared RECVF to two previous proposals for fault-tolerant CMPs and found that RECVF delivered higher energy-efficiency and lower performance degradation than either of the proposals.

Acknowledgements

We would like to thank the anonymous reviewers for their insightful comments which improved the quality of this paper. This work is partially supported by the Swedish Foundation for International Cooperation in Research and Higher Education through an institutional grant for younger researchers and by the National Science Foundation under the grant CPA-0811467.

References

[1] N. Aggarwal, P. Ranganathan, N. P. Jouppi, and J. E. Smith. Configurable isolation: building high availability systems with commodity multi-core processors. In *Proceedings of the 34th ISCA*, pages 470–481, 2007.

[2] T. Austin, V. Bertacco, S. Mahlke, and Yu Cao. Reliable Systems on Unreliable Fabrics. *IEEE Des. Test*, 25(4):322–332, 2008.

[3] Todd Austin. DIVA: A Reliable Substrate For Deep Submicron Microarchitecture Design. In *Proceedings of the 32nd MICRO*, pages 196–207, 1999.

[4] D. Bernick, B. Bruckert, P. D. Vigna, D. Garcia, R. Jardine, J. Klecka, and J. Smullen. NonStop® Advanced Architecture. In *Proceedings of DSN*, pages 12–21, 2005.

[5] S. Y. Borkar. Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation. *IEEE Micro*, 25(6):10–16, 2005.

[6] D. Brooks et al. Wattch: A Framework for Architectural-level Power Analysis and Optimizations. *Proceedings of the 27th ISCA*, pages 83–94, 2000.

[7] I. Parulkar et al. OpenSPARC: An Open Platform for Hardware Reliability Experimentation. *Fourth Workshop on Silicon Errors in Logic-System Effects (SELSE)*, 2008.

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

[9] A. Garg and M. Huang. A Performance Correctness Explicitly-Decoupled Architecture. *Proceedings of the 38th MICRO*, pages 306–317, 2008.

[10] M. Gomma, C. Scarbrough, T. N. Vijaykumar, and I. Pomeranz. Transient-Fault Recovery for Chip Multiprocessors. *Proceedings of the 30th ISCA*, pages 98–109, 2003.

[11] B. Greskamp and J. Torrellas. Paceline: Improving Single-Thread Performance in Nanoscale CMPs through Core Overclocking. In *Proceedings of the 16th PACT*, pages 213–224, 2007.

[12] C. Isci, A. Buyuktosunoglu, C-Y. Cher, P. Bose, and M. Martonosi. An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget. *Proceedings of the 39th MICRO*, pages 347–358, 2006.

[13] W. Kim, M. S. Gupta, Wei Gu-Yeon, and D. Brooks. System level analysis of fast, per-core DVFS using on-chip switching regulators. *Proceedings of the 14th HPCA*, pages 123–134, 2008.

[14] I. Koren and C. M. Krishna. *Fault Tolerant Systems*. Morgan Kaufmann Publishers Inc., 2007.

[15] S. Kumar and A. Aggarwal. Speculative instruction validation for performance-reliability trade-off. *Proceedings of the 14th HPCA*, pages 405–414, 2008.

[16] M. Kyrman, N. Kyrman, and J. F. Martinez. Cherry-MP: Correctly Integrating Checkpointed Early Resource Recycling in Chip Multiprocessors. In *Proceedings of the 38th MICRO*, pages 245–256, 2005.

[17] N. Madan and R. Balasubramonian. Power-efficient Approaches to Redundant Multithreading. *IEEE Transactions on Parallel and Distributed Systems*, pages 1066–1079, 2007.

[18] Francisco Mesa-Martinez and Jose Renau. Effective Optimistic-Checker Tandem Core Design Through Architectural Pruning. *Proceedings of the 37th MICRO*, pages 236–248, 2007.

[19] S. S. Mukherjee, M. Kontz, and S. K. Reinhardt. Detailed Design and Evaluation of Redundant Multithreading Alternatives. *Proceedings of the 29th ISCA*, pages 99–110, 2002.

[20] M. W. Rashid, E. J. Tan, M. C. Huang, and D. H. Albonesi. Exploiting Coarse-Grain Verification Parallelism for Power-Efficient Fault Tolerance. *Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques*, pages 315–328, 2005.

[21] S. K. Reinhardt and S. S. Mukherjee. Transient Fault Detection via Simultaneous Multithreading. *Proceedings of the 27th ISCA*, pages 25–36, 2002.

[22] E. Rotenberg. AR-SMT: A Microarchitectural Approach to Fault Tolerance in a Microprocessor. *Proceedings of FTCS*, pages 84–91, 1999.

[23] S. Thoziyoor et al. CACTI 5.1. *Technical Report HPL-2008-20, HP Labs*, 2008.

[24] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically Characterizing Large Scale Program Behavior. *Proceedings of the 10th ASPLOS, Oct. 2002*, pages 45–57.

[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 32nd DSN*, pages 389–398, 2002.

[26] J. C. Smolens, B. T. Gold, J. Kim, B. Falsafi, J. C. Hoe, and A. G. Nowatzyk. Fingerprinting: Bounding soft error detection latency and bandwidth. *Proceedings of the 9th ASPLOS*, pages 224–234, 2004.

[27] J. C. Smolens, B. T. Gold, B. Falsafi, and J. C. Hoe. Reunion: Complexity-Effective Multicore Redundancy. *Proceedings of the 39th MICRO*, pages 223–234, 2006.

[28] P. Subramanyan, V. Singh, K. K. Saluja, and E. Larsson. Power-Efficient Redundant Execution for Chip Multiprocessors. *Proceedings of 3rd WDSN*, 2009.

[29] P. Subramanyan, V. Singh, K. K. Saluja, and E. Larsson. Energy-Efficient Redundant Execution for Chip Multiprocessors. *Proceedings of 20th GLSVLSI*, 2010.

[30] P. Subramanyan, V. Singh, K. K. Saluja, and E. Larsson. Multiplexed Redundant Execution: A Technique for Efficient Fault Tolerance in Chip Multiprocessors. *Proceedings of DATE*, 2010.

[31] K. Sundaramoorthy, Z. Purser, and E. Rotenburg. Slipstream Processors: Improving Both Performance and Fault Tolerance. In *Proceedings of the 9th ASPLOS*, pages 257–268, 2000.

[32] E. Tune, D. Liang, D. M. Tullsen, and B. Calder. Dynamic prediction of critical path instructions. In *Proceedings of the 7th HPCA*, pages 185–195, 2001.

[33] T. N. Vijaykumar, I. Pomeranz, and K. Cheng. Transient-fault Recovery Using Simultaneous Multithreading. *Proceedings of the 29th ISCA*, pages 87–98, 2002.