Approaches to multiprocessor error recovery using an on-chip interconnect subsystem

APPROACHES TO MULTIPROCESSOR ERROR RECOVERY USING AN ON-CHIP INTERCONNECT SUBSYSTEM

A Thesis Presented

by

RAMAKRISHNA VADLAMANI

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL AND COMPUTER ENGINEERING

February 2010

ELECTRICAL AND COMPUTER ENGINEERING

APPROACHES TO MULTIPROCESSOR ERROR RECOVERY USING AN ON-CHIP INTERCONNECT SUBSYSTEM

A Thesis Presented

by

RAMAKRISHNA VADLAMANI

Approved as to style and content by:

_____
Russell G. Tessier, Chair

_____
Wayne P. Burleson, Member

_____
Sandip Kundu, Member

_____
C. V. Hollot, Department Head
Electrical & Computer Engineering

ACKNOWLEDGEMENTS

I am extremely indebted to my parents, my grandparents and my brother for earnestly encouraging me to pursue higher studies and believing in my capabilities, without which I may not have been here today.

I would like to thank my advisor, Prof. Russell Tessier and Prof. Wayne Burleson for giving me an opportunity to work in their research group and agreeing to fund me right from day one. Their unconditional guidance and critical reviews on my work, throughout my stay here, has played a significant role in the successful completion on my thesis and related publications. My special thanks to Prof. Sandip Kundu for agreeing to be on my committee and providing me with some initial insights into the implementation of my thesis work.

I would like to acknowledge the help and support from my lab mates, Jia, Deepak, Sailaja and Abhishek, with whom I have had endless coffee-time discussions and brainstorming sessions. Having teamed up with each of them at various times during my work in the group, I have really enjoyed their company and admire their illustrative work ethics.

I would like to make a special mention of the Semiconductor Research Corporation which funded my thesis work. I am also grateful to my industry liaisons for providing their valuable and timely feedback.

Lastly, I would like to acknowledge that KEB302 has had a profound impact on my life and I hope my learnings from here will bear fruits in the future.

ABSTRACT

APPROACHES TO MULTIPROCESSOR ERROR RECOVERY USING AN ON-CHIP INTERCONNECT SUBSYSTEM

RAMAKRISHNA VADLAMANI, B.E., V.J.T.I. UNIVERSITY OF MUMBAI

M.S.E.C.E, UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Russell G. Tessier

For future multicores, a dedicated interconnect subsystem for on-chip monitors was found to be highly beneficial in terms of scalability, performance and area. In this thesis, such a monitor network (MNoC) is used for multicores to support selective error identification and recovery and maintain target chip reliability in the context of dynamic voltage and frequency scaling (DVFS). A selective shared memory multiprocessor recovery is performed using MNoC in which, when an error is detected, only the group of processors sharing an application with the affected processors are recovered. Although the use of DVFS in contemporary multicores provides significant protection from unpredictable thermal events, a potential side effect can be an increased processor exposure to soft errors. To address this issue, a flexible fault prevention and recovery mechanism has been developed to selectively enable a small amount of per-core dual modular redundancy (DMR) in response to increased vulnerability, as measured by the processor architectural vulnerability factor (AVF). Our new algorithm for DMR deployment aims to provide a stable effective soft error rate (SER) by using DMR in response to DVFS caused by thermal events. The algorithm is implemented in real-time on the multicore using MNoC and controller which evaluates thermal information and multicore performance statistics in addition to error information.

DVFS experiments with a multicore simulator using standard benchmarks show an average 6% improvement in overall power consumption and a stable SER by using selective DMR versus continuous DMR deployment.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Recent high-end single and multiprocessors from Intel (Montecito), AMD (Opteron) and IBM (Cell) use extensive on-chip monitors for run-time estimates of temperature, power and performance. Specific uses of monitors to determine system critical soft-error failures, wear-out detection and security issues require fast connections on a global scale. These connections can be supported by a separate low-overhead interconnect, called monitor network-on-chip (MNoC) [1], that can be coupled to the main multicore architecture (Figure 1). Although simplified as compared to conventional network-on-chip interconnect, this new interconnect technique supports irregular routing topologies, priority-based data transfer and customized monitor interfacing that suit most on-chip monitoring applications. Collected monitor data values are manipulated by one or more processors categorized as monitor executive processors (MEPs) and the results are used to control an SoC's run-time operation. For an eight core system, the area and power overhead for the interconnection of 192 thermal monitors is less than 0.5% [1].

As the number of processors in a multiprocessor system increases, system reliability becomes of great concern. Numerous error detection and error recovery techniques (for fault tolerance) have been devised to assess processor errors and restore the correct multiprocessor system operation [2][3][4][5][6][7]. These approaches typically contain a fault control architecture consisting of an error monitor or an error detection unit, a controller for ensuring proper supervision of the fault tolerance

algorithms, restoration circuitry and inter-module interconnect. Common monitors include soft-error detectors, delay monitors, thermal monitors, and processor activity monitors. Typically, the interconnect resources used to connect monitors and controllers are simple point-to-point connections or buses. As system complexity increases, current monitor interconnections are likely to become increasingly unwieldy, encouraging the use of MNoC. Towards this end, initially this work examines the feasibility of using MNoC for supporting multiprocessor error detection and recovery. It was determined that the increasing MNoC delays or the error data injection rate for up to 32 cores lead to minimal loss of overall multiprocessor recovery performance. This motivated us to extend the use of MNoC for collaborative on-chip monitoring applications using thermal and performance monitors in addition to the error monitors.

The fault tolerant approaches based on redundancy including component dual modular redundancy (DMR) and redundant multithreading [4] may not be appropriate in all cases as they incur significant performance and power overhead and often require significant operating system support. A localized, low-overhead error reduction approach which can be selectively enabled provides a possible alternative.

In general, memory-based components in processor cores are vulnerable to single event upsets due to radiation. Although large memory structures are often protected by error checking and correcting circuits, smaller components, such as instruction queues and retirement order buffers, have less protection. Fortunately, not every bit flip in these components leads to an observable system error. A component's architectural vulnerability factor (AVF) states the probability that a fault generated in a processor structure will result in an error in the program output [8]. The AVF for

various processor structures has been shown to vary widely both across and within applications [9]. Previous studies [9][10][11] have described the efficient run-time estimation and use of AVF for single core processors in an effort to promote stable processor failure in time (FIT) rates. However, the growth of multicore use and frequent per-core voltage and frequency scaling necessitates the reexamination of AVF calculation and use.

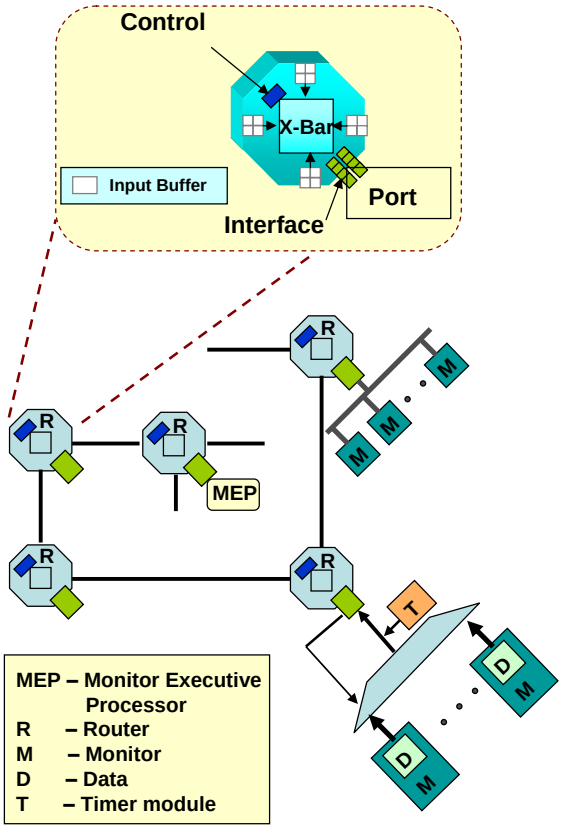

Figure 1: Detailed view of MNoC for multicores

Dynamic voltage and frequency scaling (DVFS) is commonly used in multicores to reduce hotspot temperatures and system power consumption. Unfortunately, voltage decreases and frequency increases can adversely affect system reliability [12][13], necessitating a fast system response to maintain a stable multicore soft error rate. One

approach to maintaining system reliability is to enable a small amount of redundant resources for critical system components in the presence of increased soft error risk. This risk is determined by comparing the instantaneous AVF for the components following DVFS against a predetermined threshold. If the threshold is passed, redundant components are enabled to facilitate DMR actions.

In this final experiment the power effects of using AVF-enabled DMR in a multicore environment implementing DVFS are explored. AVF values for critical resources are continually assessed throughout processing but special consideration is given following thermally-induced voltage and frequency scaling. Thermal and AVF monitor data are transported to a centralized controller via MNoC. The controller collaboratively uses the data to perform DVFS on affected cores and to enable/disable redundant resources. Our approach is designed to scale to tens of cores, enabling flexible fault coverage and performance and power control enhancement. A multicore architectural simulator and an interconnect simulator are used to assess the power and performance benefit of this approach for 8 and 16 processor multicores. An overall power benefit of 6% on average is achieved versus the continual use of redundant resources.

This thesis is organized as follows. Chapter 2 provides a detailed background related to processor error recovery, AVF, DVFS and monitor interconnects. The feasibility experiment for use of MNoC for a shared memory multiprocessor system is discussed in chapter 3 with results. Our collaborative monitoring approach, along with the algorithms and results are described in chapter 4. Chapter 5 concludes the thesis along with some discussion on future work.

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1. Errors and their detection techniques

2.1.1. Errors

A microprocessor system is susceptible to numerous types of transient and permanent faults. Transient faults include soft errors due to alpha particles, signal cross talk, and supply voltage fluctuations. Permanent faults include errors due to electromigration (wearouts) and manufacturing faults. With respect to reliability, the International Technology Roadmap for Semiconductors has predicted significant reliability problems for future systems, which will increase at a pace that has not been seen in the past [14]. Srinivasan et. al [15] showed a three-fold increase in processor wear out related faults when scaling from 180nm to 65nm. Similarly, Borkar [16] estimates a 100-fold increase in transient faults when scaling from 180nm to 16nm, while Shivakumar et. al [17] predict an even higher nine-orders-of-magnitude increase in logic circuits' transient fault rates from 1992 to 2011.

Checkpointing and rollback techniques can be used for both transient and permanent error recovery. A recently-introduced fine-grained recovery technique [18] uses rollback to counter permanent errors. A BIST-based error detection technique is used to test the output of microprocessor components such as ALUs, multipliers, and decoders. In this approach, damaged units are removed from the microprocessor datapath. The presence of multiple functional unit instances in the architecture allows for continued microprocessor operation, albeit at somewhat reduced system speed.

The checkpointing approach described in the next chapter is most appropriate for soft error recovery. Following recovery, the system re-executes checkpointed instruction on the same set of components. The availability of these components allows for continued high performance continuing forward from error recovery. The use of MNoC to identify and transport error information quickly is vital to rapid system recovery.

2.1.2. Error detection techniques

Numerous techniques have been developed to quickly identify system errors. Dual modular redundancy (DMR) uses redundant processor components, such as processor pipelines, to generate completely redundant streams of results. A result mismatch indicates an error that must be addressed. A similar software-based approach uses redundant threads. Two threads can be used to determine the same results on two different processors. A result mismatch indicates an error. A final approach compares error detection codes (CRC) for computation. The Fingerprinting [19] approach compares hashed signatures of the execution history of the processors involved in DMR to determine an error for a block of computations.

In this work, DMR is used as an error detection technique. Section 3.3 discusses our use of DMR in error checking.

2.1.3. Error containment

In a multiprocessor system, error checking and recovery can be performed at various system levels. Often, it is desirable to verify data at a specific level and prevent faulty values from moving to a higher level. For example, it might be desirable to contain a data error in an L1 cache rather than having the faulty value propagate to main

memory. The selection of the error containment level determines the amount of required checkpointing. If containment is performed close to the processor core, the amount of checkpoint storage is reduced. However, increased error detection may lead to an increased critical path length [19]. Figure 2 depicts containment at various levels. The pipeline level error detection technique used for IBM z series [20] systems reduces the impact of error checking on the critical path by performing computation and data checking simultaneously. The checkpointing scheme presented in this thesis focuses on the shared cache and the internal registers of each processor in a multiprocessor system.

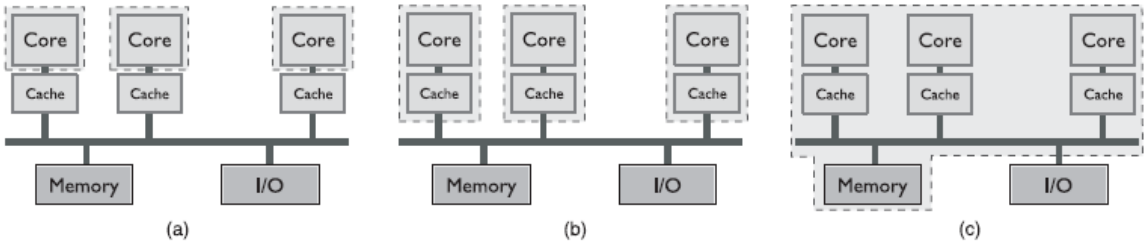

Figure 2: Illustrating error containment. (a) Core containment, (b) Cache Containment, (c) Memory containment. Adapted from [19]

2.2. Checkpointing and Rollback

2.2.1. Checkpoint

For microprocessor systems, checkpoints provide a snapshot of the architectural system state [21], including register and cache values. Frequently, copies of state values are stored in a reliable location in case they are needed later for system recovery. Checkpointing captures all information required to restart microprocessor or multiprocessor execution from a previous execution point. Usually, checkpointed information is saved in parity or ECC protected buffers that are error-tolerant. Although state data can be transferred to buffers immediately following a checkpoint, incremental

checkpointing provides a more measured approach to data transfer by slowly transferring changed values to buffers incrementally following a checkpoint. Either approach to checkpointing allows for the recovery of data which is changed following a checkpoint. A key challenge in checkpointing is system recovery speed following error detection. By introducing a low latency path from error detection to checkpoint rollback, we enhance overall system recovery speed. This approach is especially suited to real time systems where speedy recovery from a fault is extremely desirable.

2.2.2. Error Recovery

This term suggests system-level error recovery techniques. They are broadly classified into two categories: *forward* error recovery and *backward* error recovery.

2.2.2.1. Forward-error recovery

Forward-error recovery (FER) approaches attempt to identify and correct system errors through redundancy. For example, triple modular redundancy [2] of critical system components can be used to identify single faults. As the name suggests, three copies of each vulnerable component generate three copies of data. Since two of three functional units will continue to generate results which agree, normal system processing can continue unchecked simply by polling for the majority. FER systems require no checkpointing or rollback but suffer from excessive hardware overhead. As a result, FER is primarily used only in the most extreme operating environments (e.g. space exploration and military applications).

2.2.2.2. Backwards error recovery

Backwards-error recovery (BER) or rollback recovery typically uses some form of checkpointing, error detection, and rollback. The rollback process generally involves

restoring system state to a previously saved, correct configuration. Rollback is achieved by copying previously saved data to its original location in a cache, register file or memory.

2.2.3. Checkpointing and Rollback

A number of checkpointing and rollback schemes have been developed for a variety of containment levels [19] for single [21][20] and multiprocessor [2][3][22][23] systems.

A summary [22] of low overhead checkpointing schemes for backward error recovery (BER) assesses these techniques. Hardware-based checkpointing and rollback schemes can be classified using a taxonomy with three main characteristics:

1. Data error containment - This characteristic refers to the error containment granularity discussed in section 2.4. Any datum that propagates outside a system level is assumed to be correct.

2. Relative checkpoint location – This characteristic refers to the hierarchical location of the checkpointed data. *Dual* storage refers to the case when checkpoint data is stored in a location that is closely attached to the unit that is checkpointed. For instance, using a register buffer to checkpoint internal registers would fall under this category. *Leveled* storage indicates that checkpoint data is stored elsewhere in the memory system hierarchy. For instance, cache blocks or registers could be checkpointed in main memory.

3. Separation of checkpoint and active data – *Full* separation refers to the storage of checkpoint data in separate memory locations for dual storage. For example, register file checkpointing could take place in a separate physical buffer adjacent to the

register file. *Partial* separation typically involves incremental checkpointing where active and the checkpointed data are stored at the same buffer [21][3][23].

For a multiprocessor system, cache-level checkpointing and error recovery [3] can use a recovery buffer implemented alongside an L1 cache and a modified cache coherency protocol. Process checkpointing includes saving register values and flushing cache block values that have been modified since the last checkpoint. In general, cache blocks are not saved to a restore buffer immediately following a checkpoint. As cache values are modified, the original cache values are slowly migrated to the restore buffer for storage. A checkpoint counter, C_{count} is incremented every checkpoint interval if multiple checkpoints are maintained. The buffer stores copies of modified cache lines for each checkpoint. A checkpoint identifier, C_{id} is associated with each cache block. If C_{id} is less than C_{count} for a specific cache block, the block will be moved to the buffer if a write occurs. During rollback, cache blocks in the recovery buffer that are associated with a specific checkpoint are written back to the cache. The processor internal registers that were also checkpointed are reloaded and execution is restarted.

Another technique that operates in a similar fashion is SafetyNet [23]. This approach only maintains a single checkpoint and assumes about 100,000 clock cycles between checkpoints. Before a checkpoint can be completed, all multiprocessor operations must be validated as complete and correct. Once this status has been attained for all shared data values, the checkpoint identifier can be advanced.

2.3. Architectural Vulnerability Factor

As discussed in section 2.1, transient errors, including soft errors, are expected to be more frequent in future technologies. Current hardware- and software-based

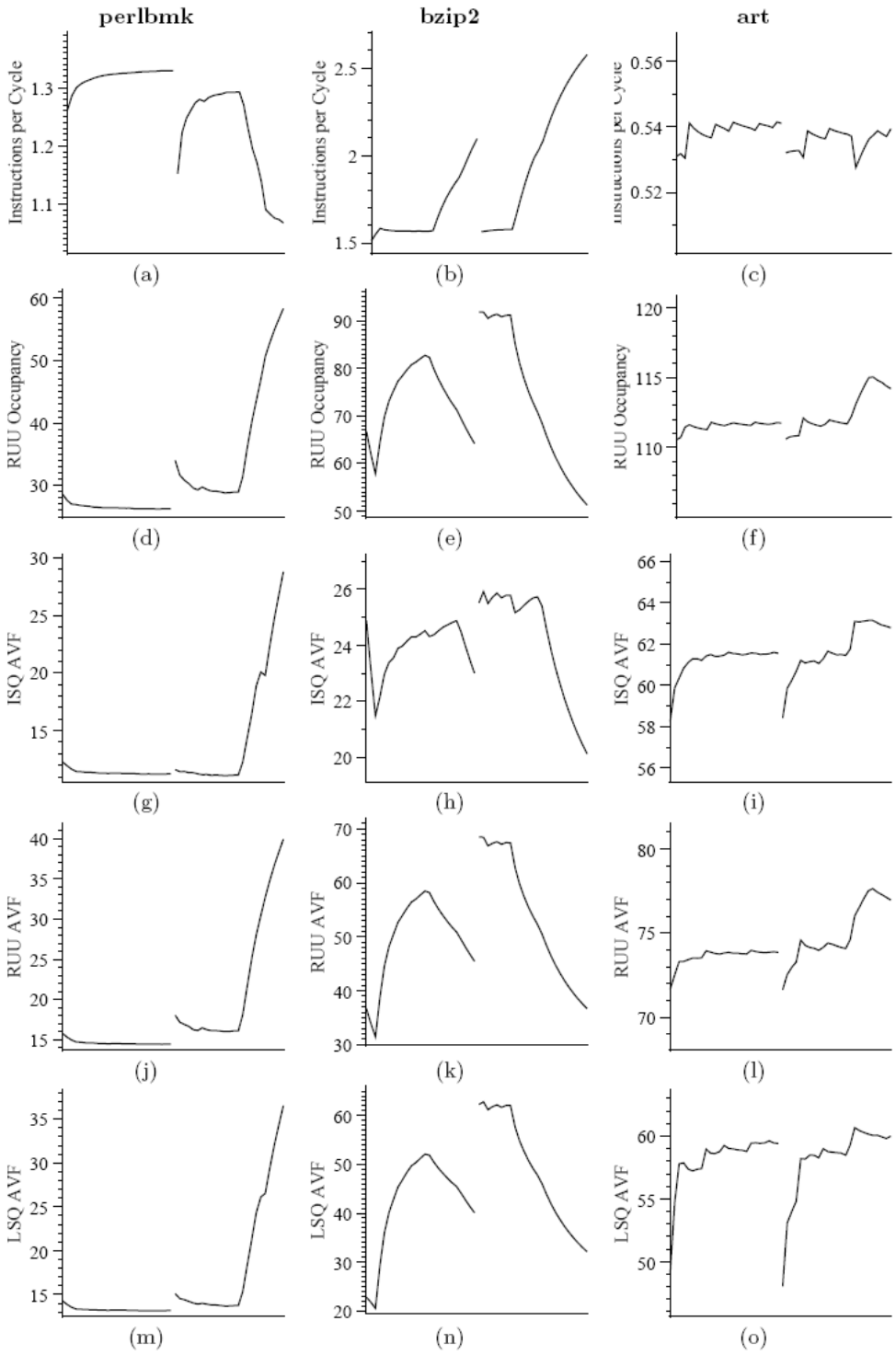

Figure 3: Time varying metrics and AVF behavior. The AVF values are along the Y-axis and time in cycles along the X-axis

redundancy techniques for implementing error detection assume a 100% probability that a given fault will manifest itself into an error and hence a failure. However, prior

work has suggested that this is almost never the case [9][11][8] and usually the vulnerability of the functional units such as the instruction queue, register file, control logic, etc to soft errors varies widely with workloads and execution time as indicated in Figure 3[9].

A *fault* is a defect in a hardware or software component. An *error* is the manifestation of a fault resulting in a deviation from the expected results. Hence, a fault can cause errors but an error may not cause faults. A fault that is masked by virtue of a program execution flow will not result in an error. *Vulnerability factor*, as defined in [8], is the probability that an internal fault in a device during its operation will result in

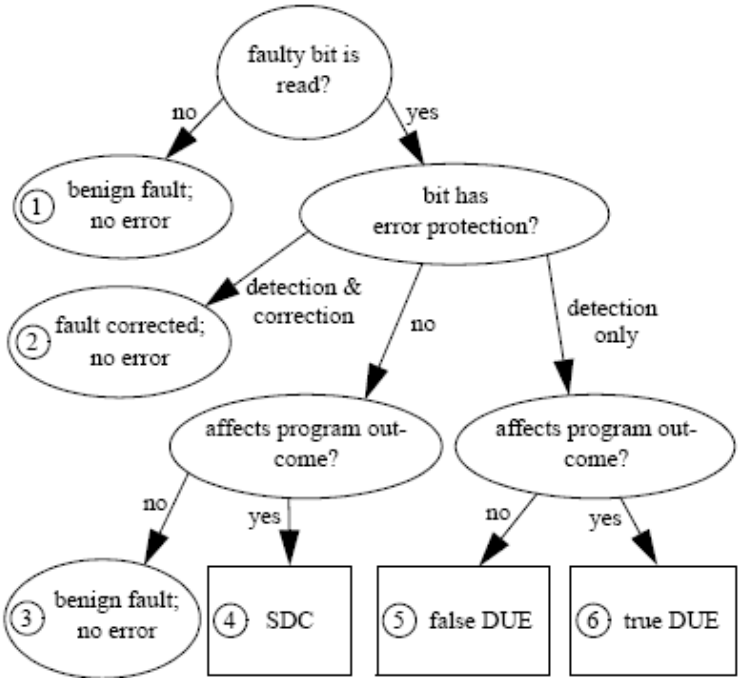

Figure 4: Classification of possible outcomes of a faulty bit in a microprocessor. SDC=silent data corruption, DUE=detected unrecoverable error. The figure is taken from [24]

an external visible error. *Failure* is caused by errors and is characterized by non-performance of expected action. A corrected error, however, does not cause failures.

Figure 4 [24] summarizes these concepts and demonstrates their interdependency clearly.

Mukherjee et. al [8] invented the term architectural vulnerability factor (AVF), which is a measure of the likelihood that a fault will convert into a visible error. Gurumurthi et al. [9] and Xiaodong et al. [10] have come up with competing online techniques for computing the AVF for several processor functional units such as the load/store queue, register file, control logic, etc. Mukherjee et al. [25] have shown that for an Itanium2-like processor architecture the AVF for the instruction queue lies between 14% and 47%, while the AVF for the execution unit lies between 4% and 27%. Similarly, for an Alpha21164-like architecture the AVF for the pipeline structure has been found to not exceed 10% [26]. These metrics suggest that affected processor units can potentially benefit from an AVF-aware redundancy scheme that disables redundant units during periods of low AVF, thus saving power [8][11][9].

Accurate run-time AVF evaluation has recently been shown to be computationally feasible [8][9]. Walcott, et al. [9] and Biswas, et al. [11] demonstrated that the aggregated AVFs of uniprocessor pipeline components can be estimated with up to a 90% accuracy using a small set of periodically-sampled microarchitectural parameters. This quantized-AVF (Q-AVF) approach is lightweight since the amount of processed data is restricted to a small quantum over a restricted sampling interval.

These approaches open up an opportunity to dynamically enable the fault tolerant redundant infrastructure *only* when the AVF is above a predefined threshold during the course of operation of the processor. Gurumurthi et al. perform a thorough *offline* simulation-based statistical analysis (involving complex calculations) for

extracting the AVF information from a set of easily-traceable processor performance metrics and use the results in an *online* predictor. This approach requires calibration for different workloads to be able to use it for a variety of real world applications. Xiadong et al. on the other hand, propose a method of estimating AVF entirely online in which, artificial errors are introduced in the functional units at a predetermined rate and the number of instances where a program failure occurred is noted to compute the AVF.

The AVF computation for a structure involves identifying the architecturally correct execution (ACE) bits (i.e. those that matter or influence the final output of a program) and un-ACE bits in that structure. Whether a bit is ACE or not depends on how a user has defined the program output. un-ACE bits are categorized as architectural and micro-architectural un-ACE bits. Examples of architectural un-ACE bits are the operand part of a NOP instruction and a prefetch instruction. If an error strikes a prefetch instruction, it will be ignored leading to a performance loss but it will not cause an incorrect execution. Micro-architectural un-ACE bits consist of the data and status bits in an IDLE state, bits in a mis-speculated state or predictor structure, etc. Thus, AVF computation for a structure (or a processor as a whole) is generally expected to be expensive and is therefore performed every several million instructions. When the combined AVF for the entire processor is known to have crossed a predetermined threshold for the past interval, the pipeline is flushed, redundant units for error detection are enabled and execution is restarted, so that the processor is protected for the next interval.

2.4. Dynamic Voltage and Frequency Scaling and AVF

AVF varies with the operating frequency and voltage of a component since it impacts the utilization of the component [27]. In Soundararajan, et al. [27], this variation was quantified for DVFS applied to a uniprocessor. More recently, Siddiqua and Gurumurthi [28] used AVF variation to support redundant multithreading (RMT) in an effort to reduce soft errors. In the latter two cases, SER levels are considered static and unaffected by per-core variations in voltage and frequency.

2.5. AVF-aware Dual Modular Redundancy

Error detection for storage components in processor-based systems is often performed using dual modular redundancy (DMR), in which outputs of duplicate copies of a component are compared before memory commits are performed [29][30]. DMR incurs a power consumption penalty and should only be used if a processor component is likely to incur soft errors. Many storage-based processor pipeline components are protected without the need for DMR. Register files and caches are generally protected by ECC/parity-check circuitry. Pipeline latches can use low-overhead error self detection and correction (i.e. Razor) [31].

Stojanovic et al. [69] came up with an ECC-protected instruction queue implementation for out-of-order processors, which has a performance overhead of less than 3% and an area overhead of the order of 10% of the size of the structure. Due to the small footprint of the additional bits and associated logic for error correction, the power dissipation is also quite less. Although our work uses DMR based error detection technique, our reliability stabilization system can work even with the ECC based

approach. Since the power consumption by the ECC-based circuits is a small percentage of the total chip power, the benefits we see may not be significant.

Additionally, the AVF of a branch predictor is always 0% since a misprediction due to a predictor soft error strike will not lead to an output error [29]. As a result, as seen in chapter 4, this work focuses on the DMR protection of specific components (instruction queue, retirement order buffer, and load store queue) which would otherwise be unprotected. The detection and rollback circuitry required to restore processor state following error detection has been discussed in detail in chapter 3 and is not described again in chapter 4.

2.6. Inter-monitor interconnection

Traditionally bus-based connections and point-to-point connections were commonly used for on-chip communication on SoC's. Velusamy et al. interfaced thermal sensors to a central controller using a bus interface [32]. McGowen et al. implemented an embedded feedback control system in which, the thermal and voltage sensors were connected to the analog-to-digital converters of the microcontroller through point-to-point links [33]. A number of error recovery techniques have been developed that propagate error data and response information through point-to-point interconnects. For instance, a recent error recovery system [34] uses clock-skewed flip flops to detect pipeline errors. Error results are individually sent to a control block that initiates instruction-retry recovery operations. An alternative approach uses area and time redundancy to improve the fault tolerance of counters [7]. In this effort, the error detection and corresponding recovery is also conducted using a point-to-point interconnect. In an integrated approach [35], an SoC resource manager architecture is

individually connected to performance and thermal monitors. The IBM Power6 monitor network [36] is composed of eight critical path delay monitors per core that are used for detecting errors during processor operation. These monitors, along with other on-chip monitors, are interconnected using a daisy chain bus.

CHAPTER 3

MNOC-BASED SHARED MEMORY MULTIPROCESSOR ROLLBACK RECOVERY SYSTEM

3.1. Introduction

In general, most multiprocessor systems in use today are based on the shared-memory programming model. These systems are frequently implemented as chip multiprocessors (CMPs), symmetric multiprocessors (SMPs), or distributed shared-memory multiprocessors (DSMs). The application of MNoC to monitors and control processors for this system represents a challenge.

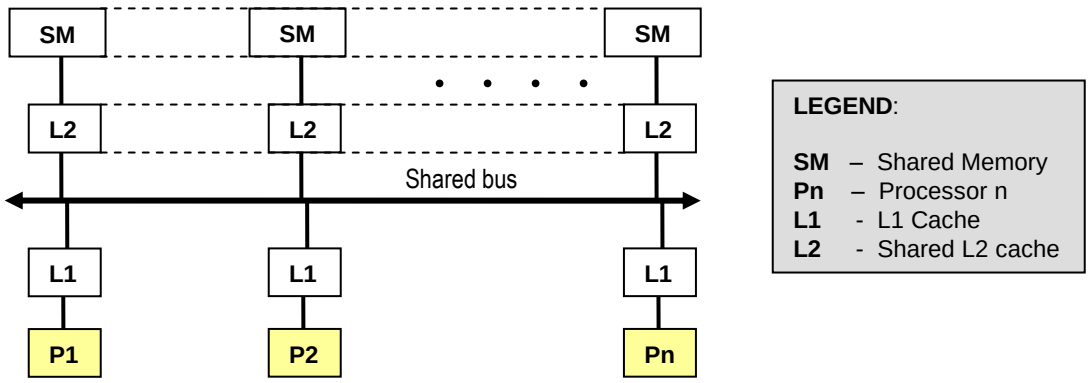

Figure 5: Shared memory multiprocessor system

Consider the shared memory multiprocessor system shown in Figure 5. For our experiment, we consider a system of 8 to 128 processing nodes, which could be used in the following configurations depending on the application under consideration:

1. Single process, multiple threads (one thread per processor)
2. Multiple processes, one thread each

3. A combination of the above two scenarios. For example, 5 cores could be included in configuration 1 and the rest of the cores could be included in configuration 2.
4. Multiple processes and multiple threads. This approach will involve context switches between processes.

3.1.1. MNoC Perspective

Scenario 3 provides an ideal configuration for testing of our monitor network-on-chip infrastructure. For scenario 3 it would be necessary to have central controller to decide which nodes need to be rolled back when one node generates an error. A low latency MNoC provides a path to quickly forward monitor data and assess a rollback strategy.

3.2. Typical Interprocessor Communication

In this example it is assumed that a single process (P), multi-threaded (T1, T2) application is running on a system configured under scenario 3. Communication has been established between threads T1 and T2 (can be extended for more processor cores). As a result of repeated computation, a data value D is written by T1 and consumed by T2. Thread T1 informs T2 of the available data value by setting a sync bit. Soon after T2 reads this data, it clears the sync bit to acknowledge the receipt of the data. Then T1 puts a new data value at D and the cycle continues.

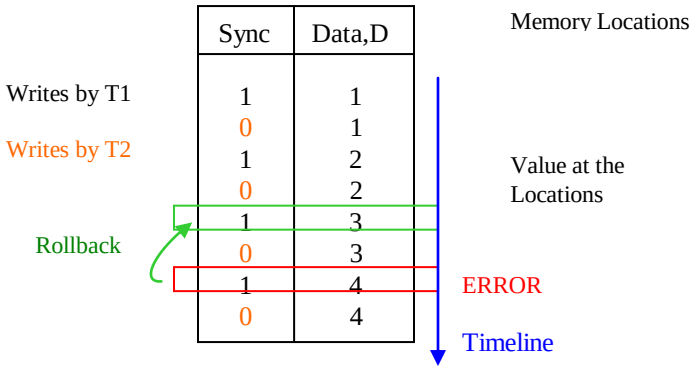

Figure 6: Typical inter-processor shared memory communication and a rollback scenario

Assume that D = 1, 2, 3, 4, 5 is written in a sequence and they have been read and acknowledged by T2 with repeated handshakes through the sync bit (note Figure 6). It is later discovered that an error was generated while processing data, D=3. At this point in time T1 is in the process of or has already generated the next data, D=4 since T2 had acknowledged the data, D=3. Hence, the system needs to roll back to a point when the memory had data value, D=3, so that T1 can roll back to an instruction that will set the sync bit and then generate D=4 and T2 can roll back to an instruction that would attempt to read D=3 from the memory.

In the next few sections we present checkpointing and rollback techniques to address the above recovery issues and also to highlight the benefits achieved by the use of MNoC.

3.3. MNoC Based Rollback Recovery Scheme

3.3.1. Architecture Description

Our MNoC-based recovery approach extends previous checkpointing methods [3][22][23]. The process of taking a checkpoint involves defining the checkpoint

interval and logging modified data in recovery buffers. The rollback process involves the restoration of logged data and the restart of computation from the saved checkpoint time. Dual modular redundancy of each processor pipeline is used for error detection; data mismatches are flagged as errors. Figure 7 provides an overview of the mechanics of the error recovery scheme.

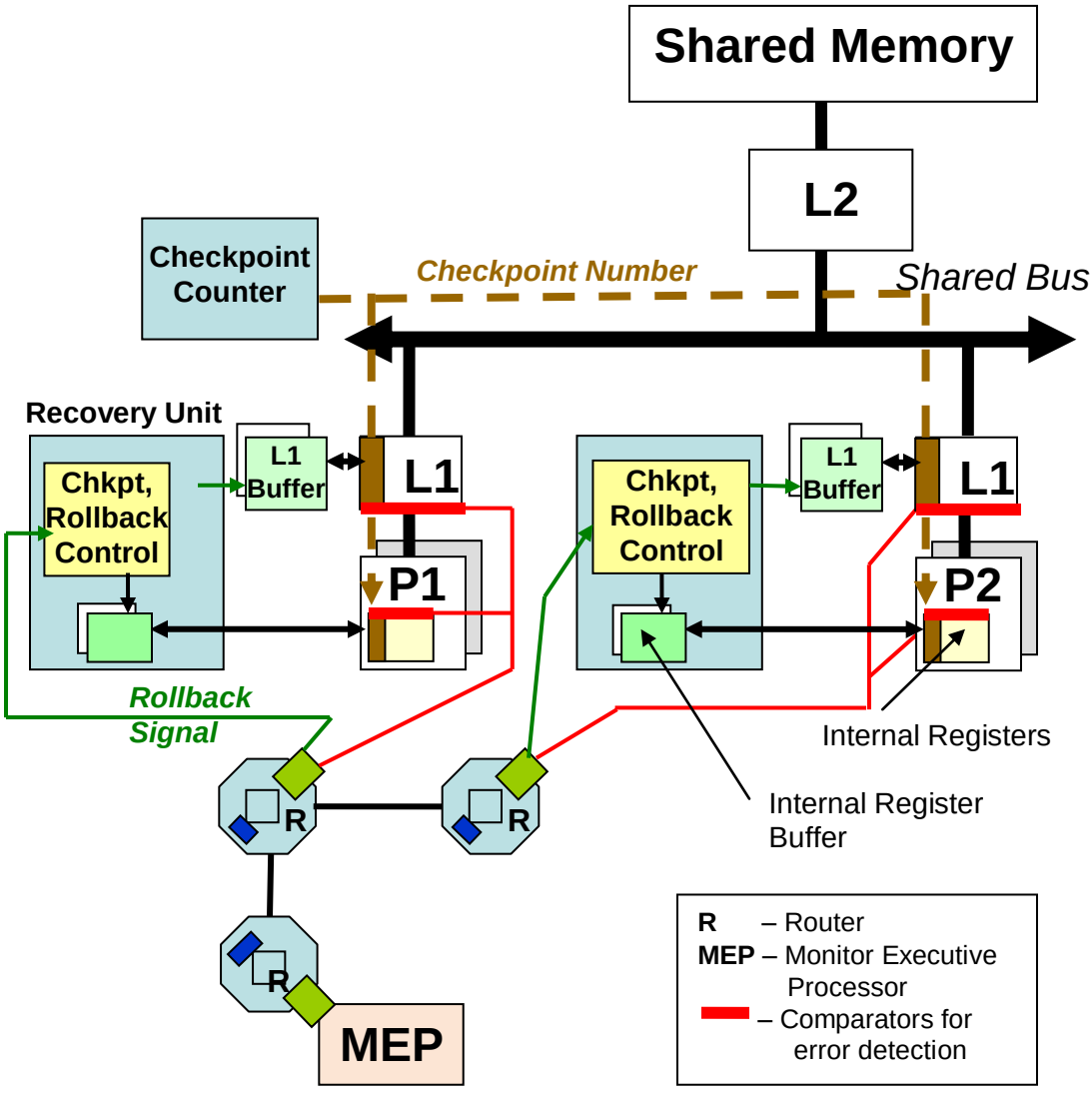

Figure 7: Error recovery scheme using MNoC

The L1 buffer holds L1 cache values that are overwritten during a checkpoint interval. In the worst case the buffer would need to be the same size as the cache. If it is

smaller, more frequent checkpoints may be required. The size of the internal register buffer for each checkpoint is equal to the number of internal registers. A checkpoint counter guides the checkpointing process. The value of the counter is incremented automatically every checkpoint interval. The number of simultaneously active checkpoints will be a research parameter.

The L1 recovery buffer is the main sub-module of the error monitoring system. If data in the recovery buffer needs to be restored, a rollback control state machine is used to coordinate the recovery. This unit conducts the checkpointing and rollback process for each node. Every node houses an error detection system to gather the error signals and transport them to the central processor, the monitor executive processor (MEP). Following processing, a rollback response message is generated by the MEP. Figure 8 illustrates the error detection system in a node.

3.3.1.1. Duplicate pipeline

For each multiprocessor node processor, the processor pipeline is duplicated and fed with the same instruction sequence as the main pipeline. All accesses to the internal register file and the L1 cache are compared with those of from the duplicate pipeline and any error is flagged. The current DMR implementation is a simplified version of the one used by the IBM G5 processor [20].

3.3.1.2. Error detection comparators

These comparators are situated at the input of the L1 Cache and at the input of the internal register file for every node in the multiprocessor system. These comparators detect any data mismatches created by the redundant processor pipelines. As shown in

Figure 8, the comparators operate in parallel with data accesses, outside of the critical path.

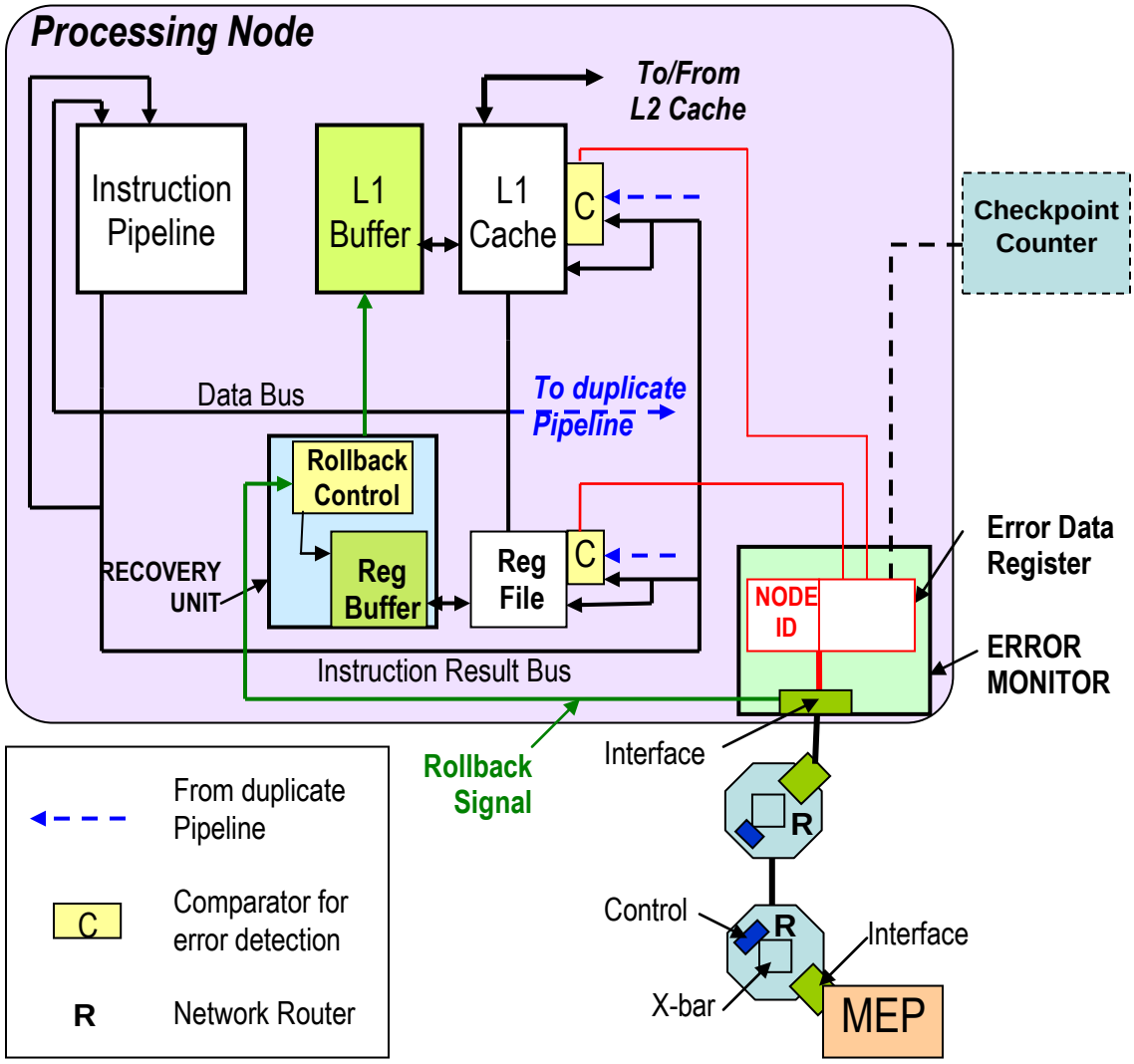

Figure 8: Error detection system using MNoC

3.3.1.3. Error Monitor

This is a lightweight module that is interfaced to an MNoC router. This module gathers error data and sends out rollback signals to associated recovery units. As shown in Figure 8, the Error Data Register (EDR) is composed of the following fields:

EDR[1:0] - the error signals from each of the two comparators, EDR[9:2] - the current checkpoint counter value, EDR[13:10] - the pre-programmed NODE ID of the node. The EDR is read through a MNoC router and is forwarded to a MEP for further processing.

NODE ID information is used by the MEP to determine the source of the error data. The MEP uses this information in conjunction with the assignment of tasks to processors to coordinate a checkpoint recovery response. Specific processors involved in the rollback are identified.

3.3.1.4. Error Data

For 16 processors, the EDR will consist of 14 bits of information. This can be scaled appropriately for up to 128 cores. To protect stored data, it is assumed that the internal register file, register buffer, L1 cache, L1 buffer, and L2 cache are all protected by error correction codes (ECC) to correct soft errors in these memory units. This feature is omitted from Figure 8 for clarity.

3.3.1.5. Incorporating MNoC

The error data generated by each node is stored in their respective error monitors. This data is transported to the nearest MEP via MNoC so that a rollback signal can be generated for the appropriate nodes in the multiprocessor system. The MNoC implementation requires a router at every error monitor as shown in Figure 8. Due to the critical nature of error detection, a priority channel is allotted for the quick transport of error data to the MEP. When the MEP receives the error data, the NODE ID is used to identify affected processors. A rollback message that consists of a rollback signal and a checkpoint number is then sent to the affected processors.

3.3.2. Checkpoint Process

As shown in Figure 7, checkpointed memory units include the L1 cache and the internal processor registers. The checkpoint counter stores a checkpoint number (CN) whose value is incremented automatically every pre-programmed interval that is at least equal to the error detection latency. Whenever a write access is made to any of the two memory units, the current checkpoint number (CN_i) is tagged to the written data. However, before the data is written, the checkpoint control checks to determine whether the previous checkpoint number tagged to the old data at that location is less than the current checkpoint number. If yes, the old data was modified in the previous checkpoint interval and should hence be logged in the buffer before new data is written to the location. After new data is written, its corresponding checkpoint number is updated to the current value in the checkpoint counter. Figure 9 summarizes this checkpointing process.

3.3.3. Rollback Process

The recovery unit initiates the rollback process in each node to restart the execution of the system from a previous safe execution point. As soon as the rollback signal is received from the MEP, the following actions are performed by the local controller.

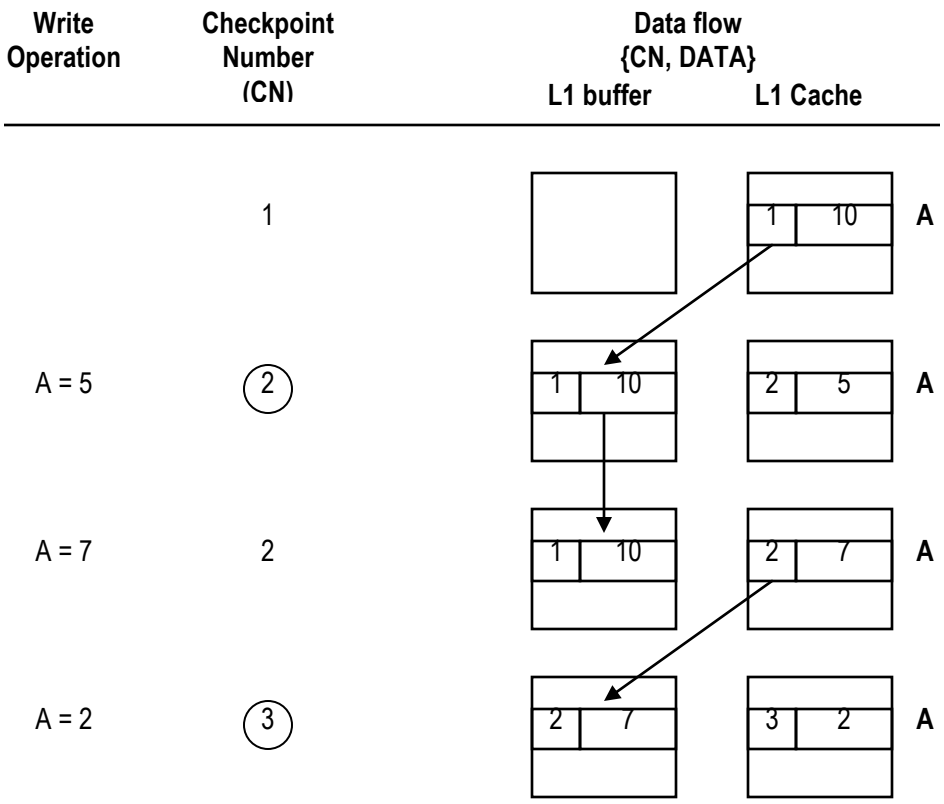

Figure 9: Illustrating the checkpoint process

Rollback Steps for the processor:

a. The recovery buffer freezes its checkpoint state and no pending instructions are allowed to update.

b. Data corresponding to the checkpoint interval is copied from the L1 buffer to the L1 cache. The remaining locations in the L1 cache are invalidated.

c. The L1 buffer is reset

d. The saved register state in the recovery buffer is restored

e. The instruction fetch is restarted.

Note that we need to store the checkpointed data for two previous checkpoint intervals. This is necessary to accommodate a situation where in the error that occurs in a previous checkpoint interval is detected in the following checkpoint interval. In this

case, the system must rollback by two checkpoints instead of one, since the last checkpoint was taken on an error data which is not desirable for restoration of the system state. Hence, in case of an error we always rollback the system by two checkpoints instead of one.

The MEP would need to send the checkpoint number to which each participating processor needs to rollback to ensure that every processor is aware of it. This may not seem necessary since the checkpoints are synchronized, however, the rollback message from the MEP to each processor may not reach at the same time due to unpredictable network delays. In such a situation the processor to which the rollback message reaches last might have just hopped a checkpoint interval. In this case that processor will not know to which past checkpoint number it should rollback.

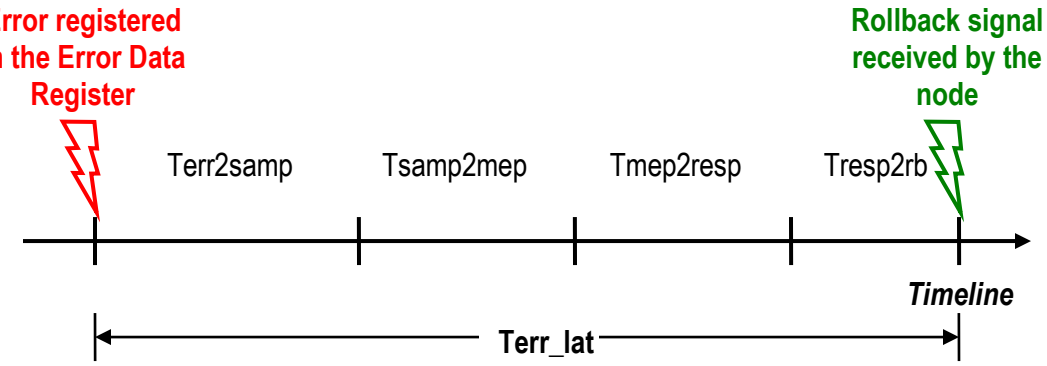

Figure 10: A closer look at the error detection latency on timeline

3.3.4. Error detection latency

Error detection latency, denoted by *Terr_lat*, indicates the time interval between when an error, as indicated by one of the error detection comparators, occurs to the time when the rollback signal reaches the recovery unit. For ease of analysis, the latency can

be split into multiple sections. Figure 10 depicts the latency on a timeline. Individual components of the time include:

Terr2samp – The time from when the error data is written to the EDR to the time when it is read by the network.

Tsamp2mep – The time from when the data in EDR is read by the network to the time when it reaches the MEP.

Tmep2resp – The time spent in the MEP to calculate the response (i.e. rollback information)

Tresp2rb – The time spent transporting the rollback signal to the recovery unit in an affected node.

3.3.5. MNoC deliberation

The timeline in Figure 10 suggests that *Terr_lat* can be reduced significantly by optimizing the interconnection between an error monitor and the MEP. This enhancement has a direct impact on the parameters *Tsamp2mep* and *Tresp2rb*. The use of MNoC reduces these values, resulting in a faster error response. Another advantage of keeping *Terr_lat* as low as possible is that checkpointing can be carried out more frequently, if necessary.

3.4. Experiments and results

The checkpointing and rollback control, checkpoint counter, L1 and register file buffers, the error detection unit and the error monitors described in section 3.3 were implemented in the SESC [43] architectural simulator. The configuration of the simulator is summarized in Table 1. Each core is assumed to contain 24 thermal, 8 delay and 1 error monitor, which are all interfaced to the MNoC in a manner described

in [1]. MNoC latency numbers for 8, 16 and 32 cores in the presence of thermal, delay and error monitor traffic were estimated using our modified PopNet simulator. These results were used in assessing the performance impact on the multiprocessor rollback recovery. Four benchmarks from the SPLASH2 suite, Ocean, Radix, Lu and FFT, were run on the multicore system, each for about 100 million cycles.

Table 1: Experimental Setup

Simulator	**SESC multiprocessor simulator**
Number of Processors	8, 16, 32
Processor Configuration	Alpha264 EV6 -like L1 $ = 32KB (private, writeback) L2 $ = 1 MB (shared, writeback) 32 Internal registers (64bit)
Soft Error Rate (SER)	1 in million cycles
Checkpointed components	L1 $, Internal registers checkpointed every 1 million cycles
Benchmarks	Ocean, Radix, LU, FFT (100 million instructions each)

3.4.1. Simulation model

The SESC multiprocessor simulator was modified to evaluate the benefits of MNoC for a multiprocessor rollback recovery system. The following modifications were implemented in the SESC simulator to better suit our experimental need.

a) Checkpoint counter: As discussed earlier, we needed a checkpoint counter to keep track of the current checkpoint interval ID and also to advance the interval periodically. The simulator was updated with this additional feature.

b) L1 buffer, internal register buffer: Section 3.3.1 discussed the role of the L1 buffer and the internal register buffer in the checkpointing process. The functionality of these buffers was incorporated into the simulator.

c) Recovery unit: As discussed in section 3.3.3, the recovery unit plays a major role in rollback recovery. The state machine that controls the checkpoint and rollback process was implemented closely with the processor core, the L1 cache and their respective buffers.

d) Error detection system: Pipeline duplication was performed for our DMR approach. Also, we modified the simulator to use a comparator at the data interface between the pipeline write back stage and the L1 cache and the interface between the write back stage and the register file. These comparators interface to the error data register in the error monitor that is connected to the MNoC.

e) MNoC: This interconnect has been modeled using the Popnet network simulator [63].

f) MEP: The MEP could be implemented as a custom state machine or we could use a dedicated processor to carry out MEP tasks. We dedicated one processor in our multicore system for the MEP.

3.4.2. MEP software flow for rollback recovery

The multiprocessor system under consideration will likely have groups of processing nodes that share tasks. For example, in a system consisting of 16 nodes, 4 nodes might run a mutually-shared application, while the next four perform a separate shared application. In such a scenario, only the nodes affected by an error require

rollback. In our experiments, the MEP software consults a static lookup table in which, each entry is loaded with the processor identification (PID) along with the ID of the application that it runs for all the cores in the system. When the MEP receives an error data (which is annotated by the NODE_ID), it is indexed to extract the application ID. The MEP then sends out the recovery to all the cores that are tagged with this application ID. The rest of the rollback steps are performed inside the individual cores as described earlier.

3.4.3. Impact of variation in MNoC delay on recovery performance

As the number of cores is increased from 8 to 32, there is a proportional increase in the number of error, thermal, and delay monitors. Due to this, the network traffic and hence the network delays increase significantly. Using the corresponding latencies offered by the MNoC for these varying workloads (as derived in [1]), the overall impact on the recovery performance has been studied. For comparison, we assume two systems, one in which MNoC is used as an interconnect for transporting the monitor data traffic to the MEP and the control action from the MEP back to the cores and another in which MNoC is replaced by a hypothetical interconnect that has a structure which is similar to MNoC but has zero latency. By this way we can accurately measure the impact of increasing MNoC delay on the recovery performance. In both the systems, checkpointing, rollback and error detection systems are active. In this experiment we assume the presence of soft errors that strike at an average rate of once in a million cycles. The graph in Figure 11 shows the performance impact for four benchmarks. The performance impact is not more than 0.1% for a 32 core system. The performance degrades with increasing MNoC delay because it contributes to

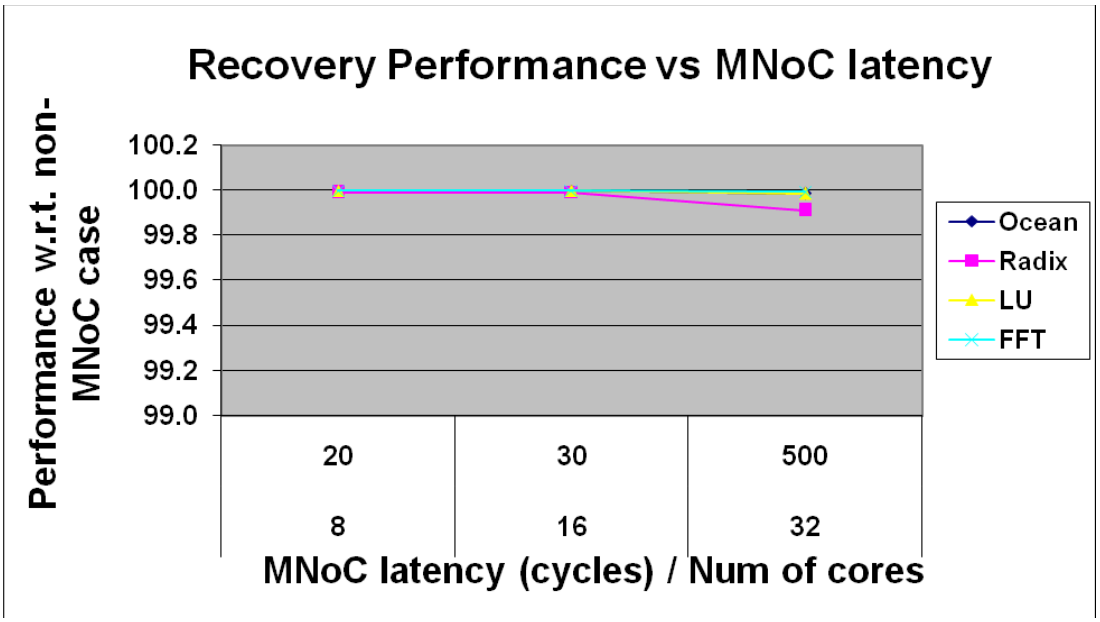

Figure 11: Impact of MNoC delay on the recovery performance as compared to non-MNoC case having a zero latency interconnect

the rollback latency and hence the rollback overhead.

The above experiment assumes the use of a regular channel in the MNoC. It was shown in [1] that the priority channel incurs a much lower latency than the regular channel, almost in the range of 20-30 cycles for up to 32 cores and above, provided the priority channel traffic is about 5% of the regular channel traffic. Since, soft errors have been shown to be very infrequent, the use of a priority channel is well justified. The impact on the recovery performance due to MNoC latency, when a priority channel is used, will be less than 0.01%. This is a 10x improvement over the regular channel MNoC case.

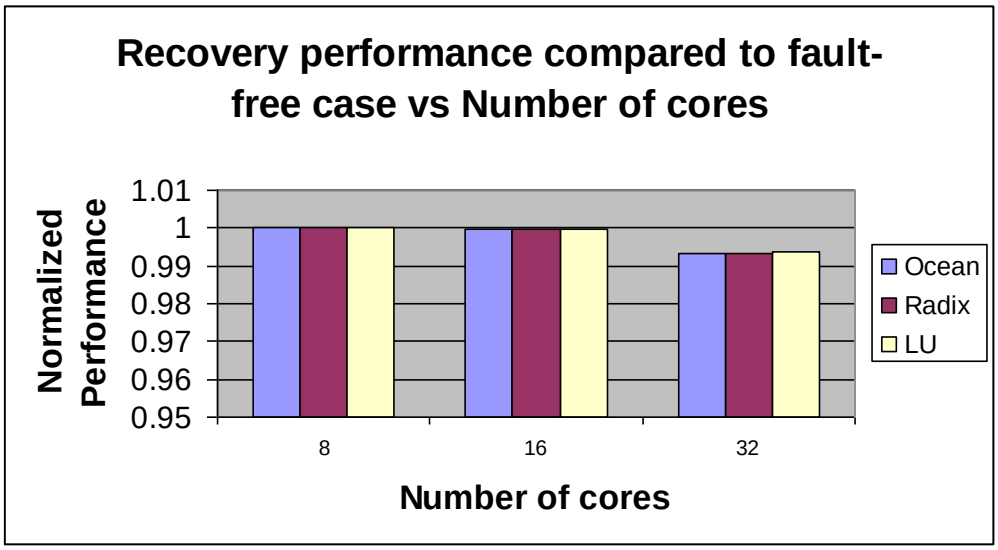

Figure 12: Performance degradation due to MNoC delay and rollback overhead

The graph in Figure 12 shows the impact of MNoC delay on the multiprocessor recovery performance when compared to a system that did not have any faults. This result helps us visualize the contribution of not only the MNoC delay but also the rollback overhead to the overall performance degradation. We see that the performance degrades by less than 1% for a 32 core system.

3.4.4. Impact of increasing error rate on recovery performance

The motivation behind this experiment is to assess the performance of MNoC when applied for errors other than soft errors which are much more frequent. For instance, delay-related errors arising due to the fluctuations in the supply voltage or voltage droops are a major concern. Similarly, dynamic voltage scaling (DVS) using delay error detection and correction is performed by reducing the supply voltage below the point of first failure (PoFF) until the error rate does not exceed 0.1% [37] or 0.04% [38] to achieve a higher overall energy gain. This experiment studies the performance of MNoC-based error recovery when employed in such relatively high error rate

scenarios.

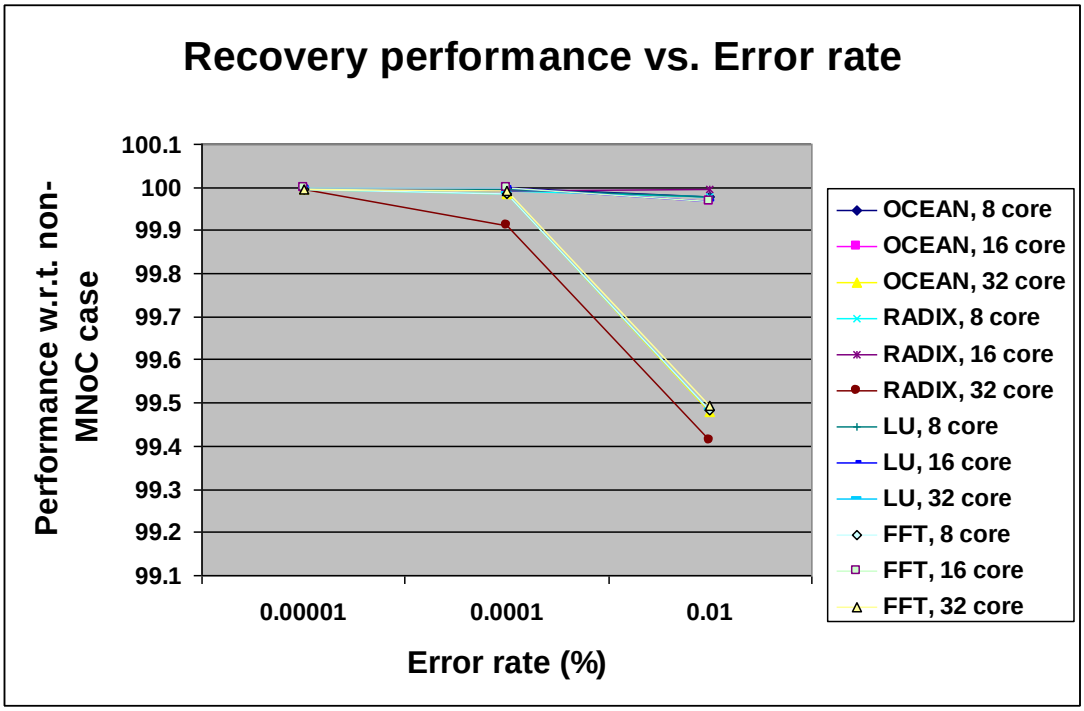

Figure 13: Impact of increasing error rate on recovery performance for a multicore system compared to non-MNoC case having zero latency

As in the previous experiment, we consider two systems, one in which MNoC is used as interconnect and the other in which a hypothetical zero latency interconnect is used. This comparison across various error rates, number of cores and benchmarks will accurately model the impact of increasing MNoC delay on the recovery performance. The graph in Figure 13 shows that for error rates as high as 0.01% (i.e. once every 10000 cycles) the impact of MNoC delay on the recovery performance for a 32 core system is less than 0.6%. The effect on system performance is noted next.

The graph in Figure 14 shows the impact of the increasing error rate on the recovery performance when compared to a system with no faults. This indicates the

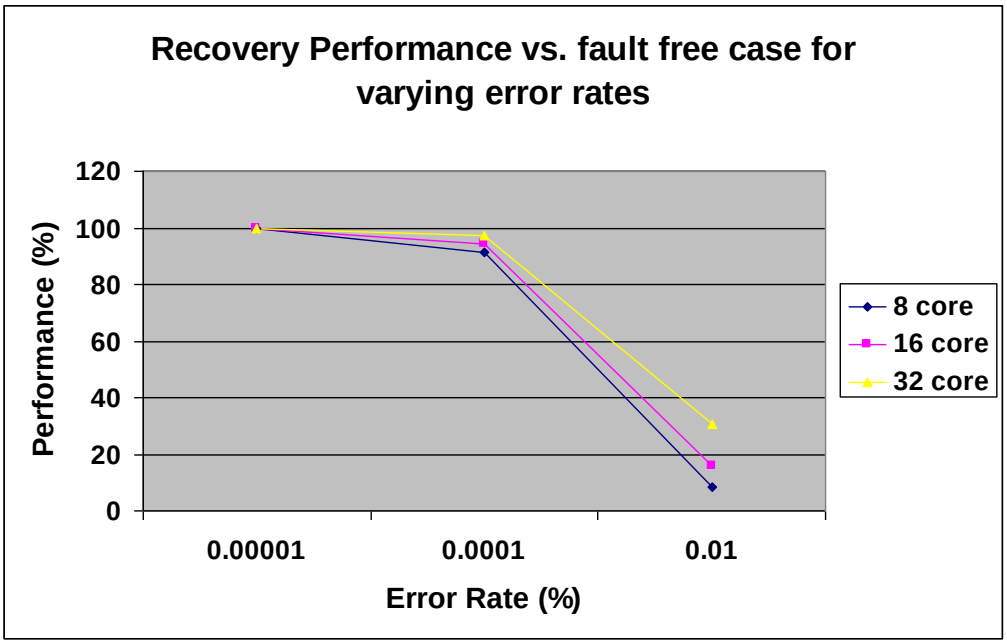

Figure 14: Impact of increasing error rate on recovery performance due to the MNoC latency and rollback overhead

contribution of the MNoC delay and the rollback overhead to the overall performance degradation. We see that the performance degrades by a significant margin as the error rate increases to 0.01% for a 32 core system.

In conclusion, the use of MNoC for various kinds of error monitors is practical. This result motivates us to look at further enhanced experiments that involve collaboration between different kinds of monitors like the architectural vulnerability monitor that could work with the error monitor in providing better overall power and performance.

CHAPTER 4

MULTICORE SOFT ERROR RATE STABILIZATION USING ADAPTIVE DUAL MODULAR REDUNDANCY

4.1. Introduction

Previous work by Soundararajan et al. [27] assesses the impact of various DVFS schemes on AVF for a single core. Hence it allows them to choose a specific DVFS scheme that optimizes the architectural vulnerability factor (AVF) the best, resulting in a reduced reliability impact. In contrast to [27] and [28], a system is developed that maintains the reliability of a multicore chip under a specified target failure-in-time (FIT) error rate while thermal-aware DVFS is performed. The reliability impact of DVFS is countered by a proportional increase in the amount of error protection in terms of increased DMR. To perform this action, a chain of relationships between the instantaneous values of voltage, frequency, soft error rate (SER) and AVF are derived to determine an optimum AVF threshold value for each processor. Once the AVF threshold is set, redundancy is enabled (disabled) when the instantaneous AVF crosses above (below) this threshold. The trend graphs in Figure 15 illustrate our idea.

Due to the masking nature of the AVF, the effective SER of a chip can be written as follows [8]:

Effective_SER = AVF * Raw_SER ---------- (4.1)

If DMR-based error protection is always provided, the effects of SER can be completely eliminated. However, for a given SER target, if AVF information is exploited, it is not necessary to provide error protection 100% of the time. Instead, it

suffices to provide protection to a structure only when the AVF of that structure exceeds a predetermined threshold. Thus we can imagine the term AVF in equation (4.1) to be an AVF threshold since any errors occurring when AVF > AVF-threshold are corrected by the protection that becomes enabled.

Effective_SER = AVF-threshold * Raw_SER ---------- (4.2)

Our goal is to maintain the Effective_SER constant as required by the chip specifications in the presence of a DVFS scheme. AVF is the probability that a bit contributes to the final output of the program. This behavior is completely dependent on

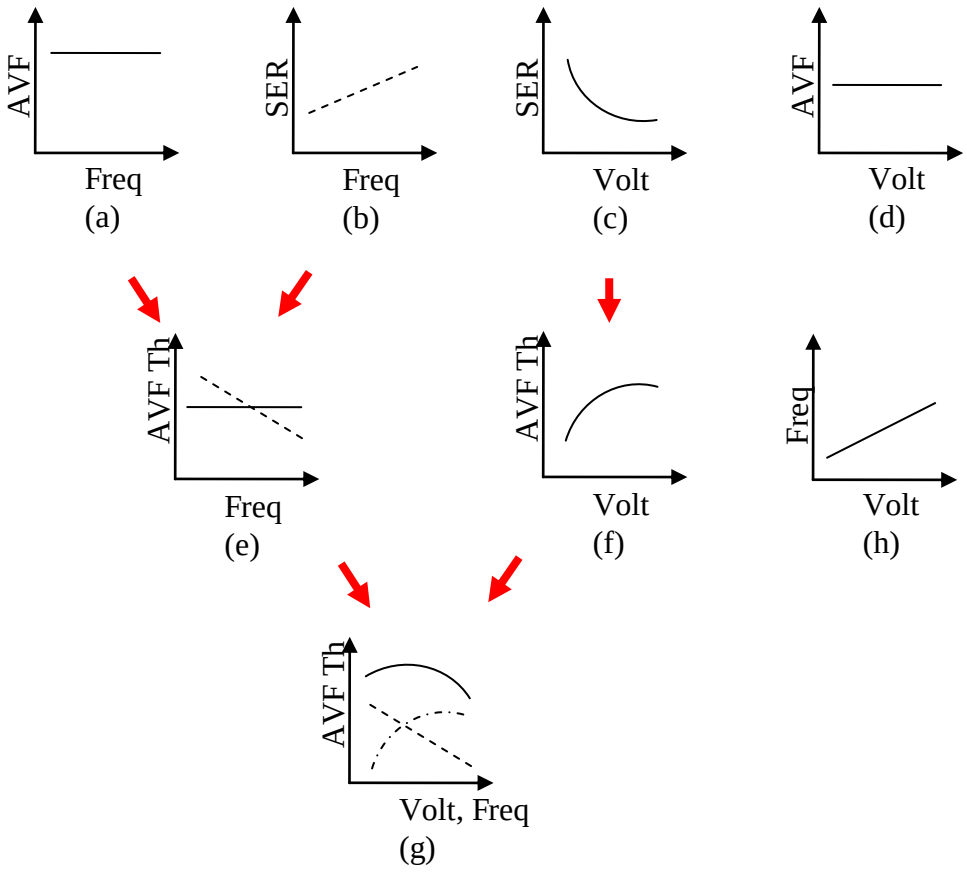

Figure 15: Conceptualization of collaboration between AVF, voltage and frequency information from across multicores to dynamically arrive at an AVF threshold value. Refer eq. (1) and (2) as well.

the program behavior and flow. Since a program execution is cycle dependent, AVF seems to be independent of frequency of operation of a processor as shown in Figure 14 (a). Referring to Figure 15 (d), AVF does not change with supply voltage [27]. Since AVF is based on the utilization of a structure, the processor operating frequency affects the instruction flow rate through the structure and changes its AVF [27]. SER has been shown to increase in proportion to the operating frequency and reduce exponentially with the supply voltage [39]. In general, the variation in a structure's AVF due to frequency or other parameters will not impact the AVF threshold, since the presence of a threshold will compensate for the increased AVF by enabling redundancy more frequently. However, a change in the raw SER due to temperature or voltage variation (Figure 15(b), (c)) requires a corresponding change in the AVF threshold to maintain a constant effective SER (refer to equation (4.2)). This observation forms the basis for the graphs in Figure (e), (f), (g). Hence, AVF threshold is a function of instantaneous supply voltage and frequency. The DVFS algorithm assumes that the frequency scales linearly with supply voltage as discussed in [12] and [13] (Figure 11(h)).

4.2. Adaptive AVF Calculation and Use for DMR

Our adaptive DMR approach requires real-time AVF computation and the use of an interconnect architecture for thermal monitor and system parameter data collection and processing. Three specific operating scenarios are considered in which real-time AVF information is used to enable/disable component-based DMR for the instruction queue (IQ), retirement order buffer (ROB), and load-store queue (LSQ):

1. AVF information is used to enable/disable DMR for the components which exhibit an AVF below a predetermined, fixed threshold.

2. AVF information is used to enable/disable DMR for the components which exhibit an AVF below a dynamically-determined, variable threshold which changes with voltage and frequency updates.

3. AVF information is ignored and DMR is always enabled for the components.

Each of these cases is considered in the context of multicore DVFS performed in response to thermal events.

4.3. Disabling DMR Components

Power gating and clock gating are two common procedures to reduce the dynamic and static power consumption of processor structures.

Power gating involves disabling the header transistor in the gates that help reduce the leakage in addition to overall power. Hence, this technique has an associated timing overhead. Hu et al. [67] have discussed this overhead in detail. Homayoun et al. [68] have discussed the potential of power gating for instruction queue in a superscalar processor, since this unit is usually responsible for 27% of a superscalar processor. Their technique has been shown to reduce up to 95% of leakage power during idle times.

Clock gating has a comparatively lower timing overhead since it involves gating of the clock supply to a module rather than gating all the gates in the block [44]. This does not reduce the leakage power to a large extent. As a result, the power savings are expected to be lower than the power gating approach.

Our work focuses on power savings of our variable AVF threshold approaches. Hence, we use power gating for the unused redundant resources (IQ, ROB or LD/ST

queue) in each processor and assume that the resulting performance overhead is tolerable.

4.4. AVF Computation in a Multicore Environment

AVF calculation for IQ, ROB, and LSQ components must occur periodically since AVF values typically show significant run time variation [9][29]. The AVF of each component is determined using microarchitectural parameters obtained from the processor. A linear combination of eight parameters can be combined [11] to describe the AVF for each component at an accuracy level approaching 90%. These parameters include:

1. Stores flushed before data translation lookaside buffer response.
2. Store buffer utilization
3. Retirement order buffer empty cycles
4. Retirement order buffer utilization
5. Branch misprediction count
6. Reservation station utilization
7. Instruction queue utilization
8. Total front-end instruction kill latency

Each parameter is scaled and linearly combined to form the AVF estimates for the three components (IQ, ROB, LSQ) in each processor. Since our processor model is slightly less complex than the one used in [11], the coefficients for parameters 1, 6, and 8 are set to zero. The five remaining parameters related to AVF calculation are commonly monitored in hardware in microprocessors. The required performance

monitoring hardware often consists of two parts [40], an event detector and an event counter.

To evaluate AVF in our system, fixed event detectors and counters are needed to collect desired performance information. For example, store buffer (STB) utilization can be determined by monitoring store buffer write and read events. A corresponding counter for STB utilization increases by 1 when an STB write occurs and decreases by 1 when an STB read occurs. Since the size of the STB is known, it is straightforward to calculate STB utilization from the counter. The same method works for the utilization calculation of an ROB, a reservation station and an instruction decode queue. Event detectors are connected to both the read and write signals of the target structure. A write event increments the counter and a read event decreases the counter for the target structure. To determine ROB empty cycles, an event detector is connected to the ROB empty signal. A count is incremented for each cycle the signal indicates an ROB empty. The similar method works for the branch misprediction counter. The event detector is connected to a misprediction signal. Whenever a misprediction happens, the counter increases by 1.

Figure 16 shows the structure of a processor pipeline and the associated AVF monitoring circuitry. Event detectors are connected to the IQ, LSQ, ROB and branch predictor to probe operations in these units. Some detectors have been omitted from this figure for clarity. Five counters are connected to the corresponding detectors to obtain utilization information for the five parameters. The hardware cost of the performance counters and detectors is modest. AVF calculation is performed every 1024 cycles [11]

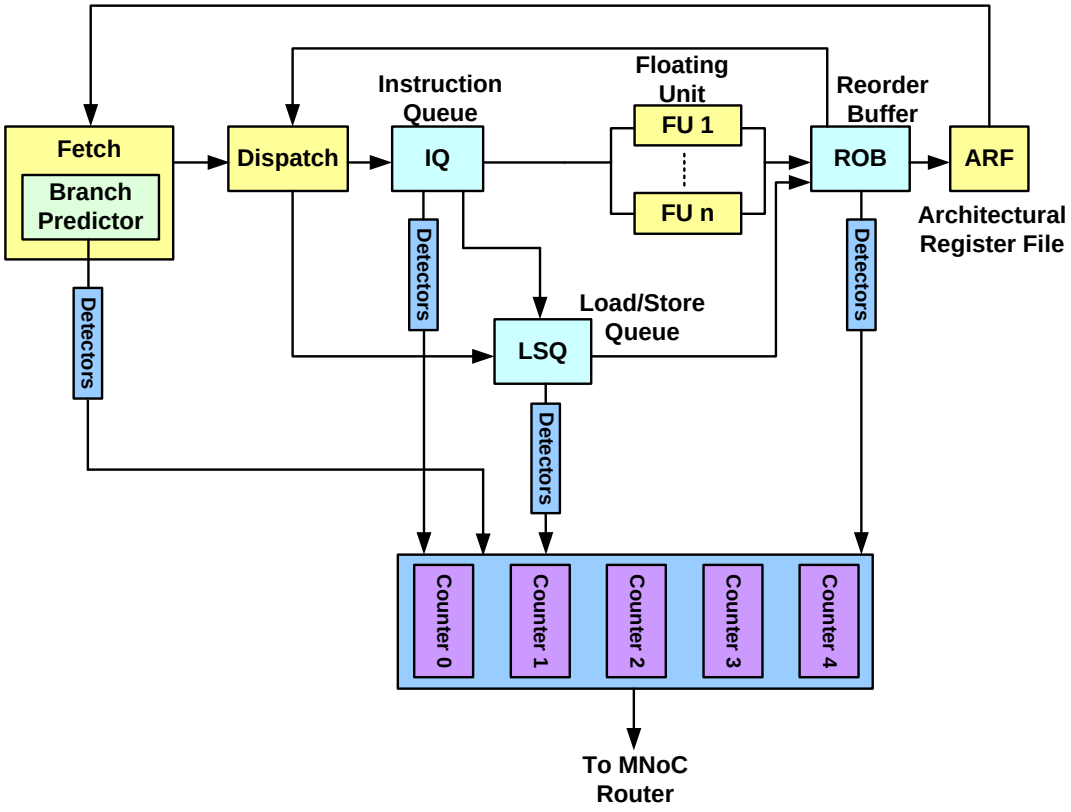

Figure 16: AVF monitor for one pipeline

leading to a counter requirement of 5 * 10 bits = 50 bits. Each detector can be implemented in a small number of logic gates.

The hardware overhead required to duplicate the IQ, LSQ, and ROB is also modest. For our architecture, based on an Alpha264, a total of 16, 112 and 176 thirty-two bit values are needed (see Table 2). This analysis indicates a total of 9728 storage bits. As a result, the total hardware overhead for per-core AVF-enabled DMR is about 200K transistors, a small percentage of the total processor transistor count, including cache.

As shown in Figure 16, monitored information is transferred to a centralized processor using an interconnect network (MNoC), which is discussed later in this

chapter. The centralized processor (MEP) calculates the AVF of each core based on the obtained counter values.

4.5. Reliability-aware AVF threshold computation

Processing components require a stable SER to operate properly. Due to the masking capability of the AVF, the effective SER of a processor core [8] is defined as:

Effective_SER = AVF * Raw_SER (4.3)

The Raw_SER (total expected bit flip rate) is reduced to an Effective_SER since not all soft errors eventually affect the visible program output. Processors generally have a target effective_SER threshold (Target_SER) which is predefined for the architecture. To ensure proper operation, it is desirable to keep the instantaneous SER of the processor core components below the target SER. If the rate rises above the target SER threshold, resource redundancy can be used to mitigate errors. Equation (4.3) can be rewritten as:

Target_SER = AVF_threshold * Raw_SER (4.4)

which indicates that the target SER threshold is directly related to the AVF_threshold. If the Raw_SER is constant, the need for component redundancy can be directly determined from the measured AVF. A measured value for a component which is above the AVF threshold indicates the need for DMR. Otherwise, DMR can be deactivated.

$$SER(V, f) = SER_0 \cdot 10^{\frac{d(1-x)}{1-f_{\min}}} \quad (4.5)$$

In most cases, however, Raw_SER is not constant. For example, the SER fault model in Equation (4.5) [12][39] assumes an exponential relationship for SER with respect to the frequency and supply voltage. In the equation, f_{min} corresponds to a normalized minimum-energy frequency [41] (e.g. an f_{min} of 0.2 indicates the minimum frequency is 20% of the maximum) and x indicates the relative scaling of f and V between their min and max values.

Frequencies below f_{min} (typically 5% of f_{max} [41]) consume additional energy due to increased memory latency. Parameter d = 2 is based on the expected fault injection source [12]. SER0 is the raw SER corresponding to the maximum voltage and frequency used by the multicore. This model indicates that both AVF_threshold and instantaneous SER must be considered in determining the need for DMR. If raw SER increases, the AVF threshold used to enable redundancy must be reduced so that target SER levels are not crossed. In summary, the relationship between AVF_threshold and Target_SER can be expressed as:

AVF_Threshold = Target_SER/SER(V,f) (4.5)

where SER(V, f) can be calculated using Equation (4.4).

4.6. Example AVF threshold and overhead computation

The AVF thresholds used for experimentation in this work were determined as follows. Mukherjee et al. [8] determined a FIT of approximately 200 to 2000 for the 200,000 vulnerable bits in the SPARC64 microprocessor. Since each core (based on an Alpha264) in our example multicore has roughly 10% the number of vulnerable bits of a SPARC64, raw_SER in Equation (4.3) is set to 28 FIT. Additionally, Mukherjee et al.

determined a 1000 year mean time between failures (MTBF) for a SPARC64 and suggested that MTBF should be increased proportionally for each core when a multicore system is considered. For our 8 core system, an MBTF of 8000 years is used to achieve an overall 1000 year MTBF. This MTBF corresponds to a 14 FIT per core (109/8000*365*24), which is our target_SER. Using Equation (4.4), an AVF_threshold of 50% is determined.

Our system dynamically computes AVF thresholds for each core in the system based on current frequency and voltage values. A two-level DVFS system is implemented which switches the voltage and frequency between more aggressive (2GHz, 1.2V) and less aggressive (1GHz, 0.84V) parameters. Since f_{min} for our system is equal to 0.05 * f_{max}, f_{min} is set to 100MHz/2GHz = 1/20 after normalization. When this value of f_{min} and the raw_SER (i.e. SER0) of 28 FIT are used in Equation (4.4), a new raw_SER at 0.84V, 1GHz is set to 10 times SER0. Using Equation (4.5), this value leads to an adjusted AVF_threshold of 25% for low voltage (0.84V). This adjustment can be applied since AVF values are influenced by frequency changes [27].

4.7. Experimental Approach

4.7.1. Monitor Network on Chip

Our AVF-enabled DMR approach benefits from the use of a dedicated interconnect for monitor traffic. The five microarchitectural parameters listed in section 4.3. are collected by an AVF monitor located in each processing core (Figure 16). The monitor interfaces to lightweight monitor network on chip (MNoC) routers which transport the information to a centralized monitor executive processor (MEP) [1]. The

irregular topology support provided by MNoC is suited to the distributed placement of an AVF monitor and one MNoC router in each core (Figure 17). In addition to AVF information, MNoC also transports thermal monitor information and control

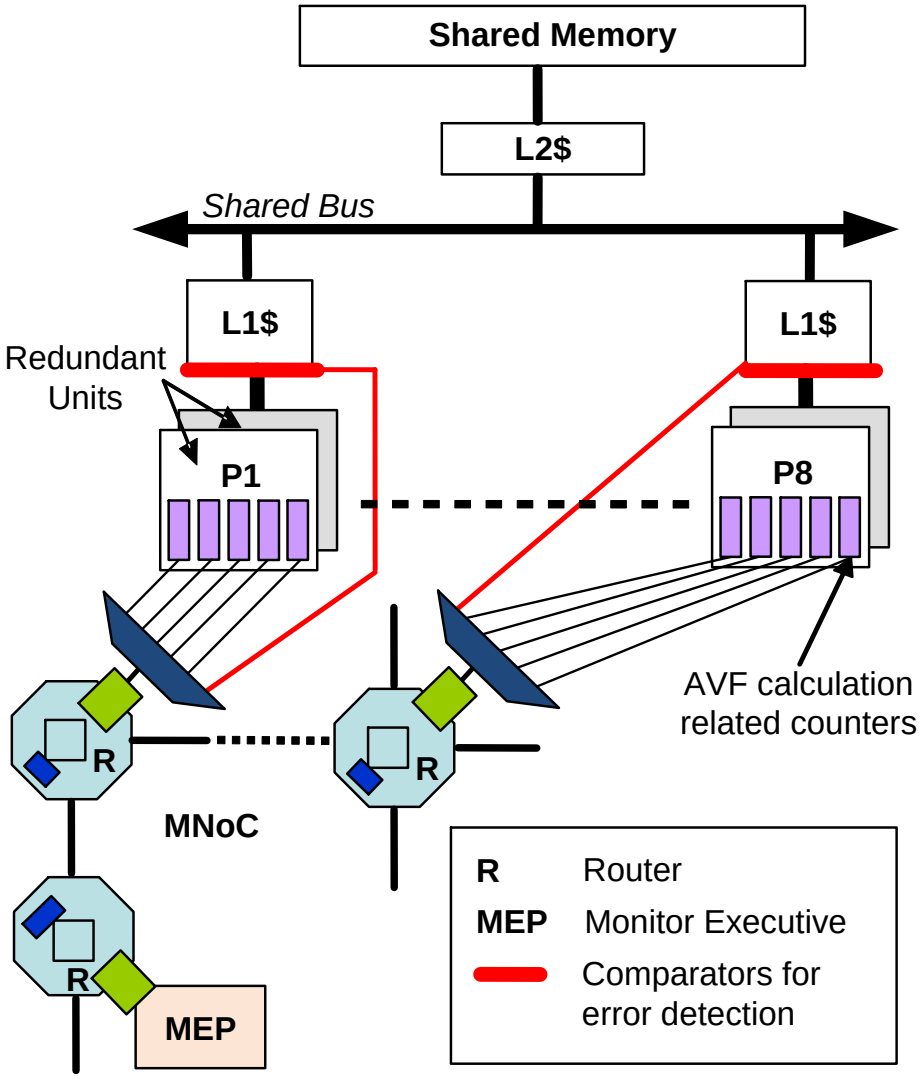

Figure 17: An 8-core system for AVF-aware DMR throttling

information which modulates per-core voltage and frequency. A total of 8 thermal monitors are allocated per core to achieve 0.1 degC accuracy [1].

4.7.2. AVF-aware DMR throttling

Shared memory multiprocessor systems consisting of 8 and 16 cores are used for experimentation. Each processor contains the following duplicated pipeline structures: instruction queue, retirement-order buffer/reservation stations and load/store buffers. When DMR is enabled for a pipeline component, a per-core error detection system flags an error if a component output does not match the data from its counterpart. An AVF monitor, the 8 thermal monitors, and 3 error monitors per core are connected to an MNoC router via a multiplexer. The error monitor's low bandwidth limits their impact on monitor data and processing.

The MEP executes the following DMR throttling algorithm for each redundant component in each core. Initially, all redundancy is enabled. Each AVF monitor in the multiprocessor system is then sampled in a round-robin fashion. When the AVF values for a processor component falls below a predefined lower threshold, the MEP sends a disable signal for the replicated resource. If the AVF is greater than the threshold, the redundant resource is re-enabled. Sequentially, the DVFS algorithm on the MEP proceeds as follows:

1. Measure the instantaneous values of temperature (T) and AVFs for each core in each sampling period

2. If T > threshold, reduce V, f (perform DVFS) for affected core. Update AVF_Threshold values.

3. If AVF > AVF_Threshold for a processor component, enable DMR for the component. Otherwise disable DMR.

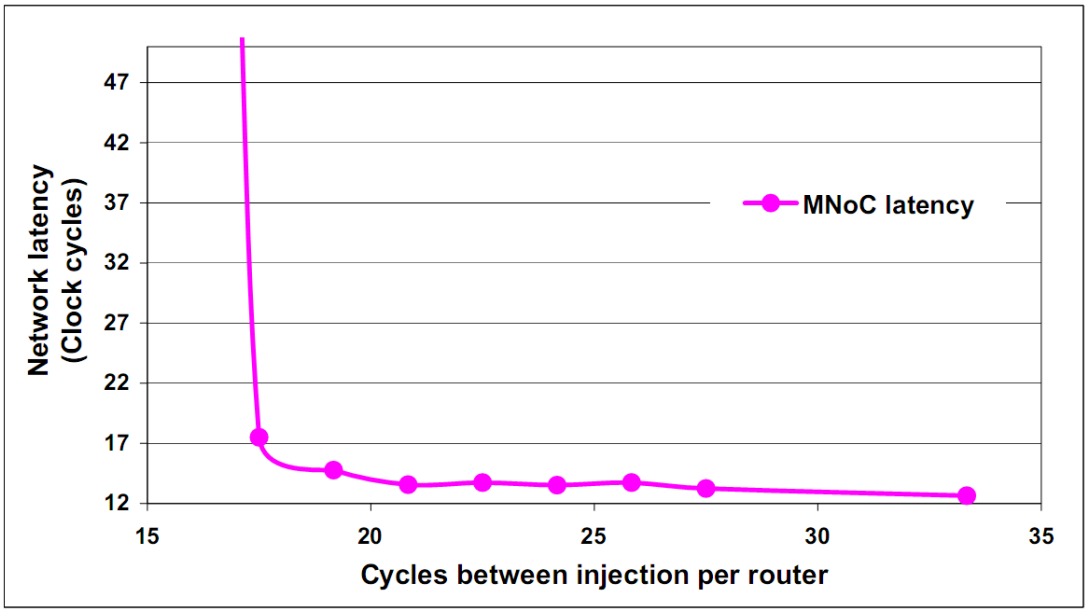

Figure 18: Latency vs. injection rate per router for a 16 router system. Results were generated using a modified Popnet simulator

The Update_AVF_Threshold routine required in the algorithm uses Equation (4.5).

4.7.3. MNoC Performance and Interface to AVF monitor

Based on previous work [42], the sampling rate of a thermal monitor is set to 1 per 800 clock cycles at 500 MHz. The five architectural parameters (discussed earlier) needed for AVF calculation are transferred to the MEP once every 1024 system cycles. Since one MNoC router is used per core, the injection rate per MNoC router is 8 * (1 per 800 cycles) + 5 * (1 per 1024) = 1 injection per 67 clock cycles.

To assess the expected MNoC monitor data latency in MNoC, a latency experiment was performed for a 16 processor multicore system. As shown in Figure 18, as long as the injection rate per router is less than 1 per 17 cycles, MNoC latency remains consistently low. In our system, the injection rate per MNoC router is low

enough to provide an MNoC latency of less than 15 clock cycles. For 8 core multiprocessors, our experiments show a similar 15 cycle latency.

4.7.4. Experimental Procedure and Results

The simulation setup including the configuration of the simulated shared memory multiprocessor system is summarized in Table 2. A modified SESC [43] multiprocessor architectural simulator is used to evaluate the run-time effects of DVFS on a series of applications and the collection of information from one AVF (Figure 16) and 8 thermal monitors in each processor core. The MEP functionality is assigned to one of the cores in the simulated multicore system.

The processor power model used by SESC is based on Wattch [44]. The cache power model is based on CACTI [45] and the temperature model for both (called SESCSpot) is based on HotSpot [42]. SESCSpot calculates the temperature of processor subblocks based on the power trace of the architecture in a post processing fashion. The processor architecture is modeled on an Alpha264 with a MIPS ISA and the floorplan of each processor core used for thermal modeling is based on prior work [46]. For our DVFS implementation we integrated SESCSpot into the core of the SESC simulator to obtain temperature readings at run-time. This approach enabled the MEP to sample the temperature readings at run-time and execute the DVFS algorithm.

In order to assess the benefits of our AVF-based dual modular redundancy approach, the three specific operating scenarios are considered:

1. AVF threshold fixed – DMR enabled when a component AVF passes a fixed threshold

2. AVF variable threshold – DMR enabled when a component AVF passes a threshold which varies with DVFS based on Equation (4.5).

3. Full DMR: DMR is always enabled for all three components (IQ, ROB, and LSQ).

All three of these cases are considered in the context of DVFS. The third case is the worst case scenario and it is used as a baseline for the other two. The first case

Table 2: System Setup

Simulator	SESC multiprocessor simulator
Technology	90 nm
Num of processors	8, 16
DVFS V, f levels	f(high)=2GHz, V(high)=1.2V f(low)=1GHz, V(low)=0.84V
Benchmarks	SPLASH2 (400M instructions each)
Processor configuration	
Instruction Issue	4 out-of-order
I-cache	64KB, 4-way
D-cache	64KB, 8-way, 2 cycles
Branch Predictor	Hybrid
Branch Target Buffer	4K entries, 16-way
Instruction Queue	16 entries
Retirement Order Buffer	176 entries
Load/Store Buffers	56/56 entries
L2 Cache	1MB, 8-way, 10 cycles

considers the AVF threshold for a component to be fixed regardless of voltage and frequency. As a result, the AVF threshold must be set to a reduced value of 0.25 (25% of bits are important) which is used during both high voltage and low voltage usage. The second case considers the AVF threshold as dynamically varying as DVFS changes

Table 3: Power benefit and overhead results for 8 and 16 core system

Test bench name	Case	8 core		16 core	
		Power per core (W)	**Power benefit (%)**	**Power per core (W)**	**Power benefit (%)**
LU	Full DMR	11.50		11.75	
	Fixed threshold	10.88	**5.39**	11.19	**4.77**
	Variable threshold	10.80	**6.09**	11.10	**5.53**
Ocean	Full DMR	9.83		10.04	
	Fixed threshold	9.63	**2.03**	9.63	**4.08**
	Variable threshold	9.13	**7.12**	9.29	**7.47**
FMM	Full DMR	14.28		10.28	
	Fixed threshold	14.13	**1.05**	9.75	**5.16**
	Variable threshold	12.28	**14.01**	9.69	**5.74**
Radix	Full DMR	4.48		4.25	
	Fixed threshold	4.38	**2.23**	4.13	**2.82**
	Variable threshold	4.12	**8.04**	3.94	**7.29**

voltage and frequency levels. AVF thresholds of between 25% and 50% are determined by the MEP for each processor component.

The power benefits of a variable AVF threshold in enabling DMR are shown in Table 3 for four SPLASH2 benchmarks mapped to 8 and 16 cores. Portions of each benchmark are distributed across the cores. DMR is only performed on the specific processor components which have an AVF greater than the target threshold. On average, the variable AVF threshold approach (case 1) reduces core power (without cache) versus full DMR (case 3) by about 8% and 6%, respectively, for 8 and 16 core processors. An average power improvement of 6% and 2% is seen for the variable AVF threshold approach versus the fixed AVF threshold approach. In general, the cost of providing a stable SER through DMR is low. The power cost of including DMR is about 5% for 8 cores and 6% for 16 versus unprotected scenarios. The power

consumption of MNoC (~250 mW) is considered in these calculations. The 8-core FMM application shows a particular savings with a variable versus fixed threshold (14% vs. 1%) since most AVF values are above the fixed threshold.

Even though the precision of Wattch has not been discussed explicitly, it has been shown to have an accuracy of 10% and a relative accuracy of 10-13% [44]. Our power consumption results have been generated using Wattch. Since we measure the relative benefit, precison of Wattch is an important factor in determining the reliability

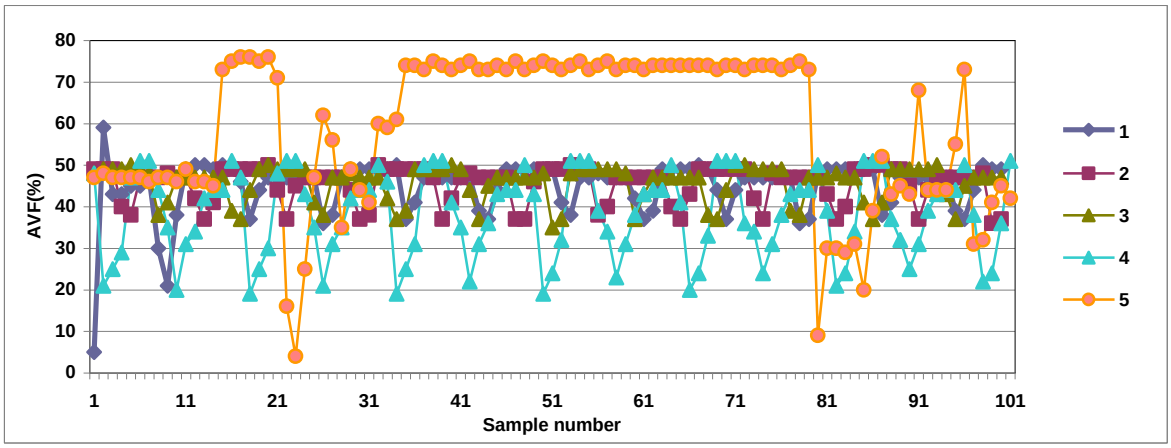

Figure 19: Five AVF traces (Y axis) for an instruction queue across 100 consecutive sampling intervals (X axis) for the LU benchmark

of our results.

The variability of AVF is apparent from Figure 19, which shows AVF variation across LU benchmark run time for an instruction queue for five traces of 100 samples. AVF values are measured over several time trials. In general, calculated AVF is mostly at or below 50% with frequent deviations over a wide range.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

As the process technologies scale and the multicores begin to gain a commonplace status in the processor based systems, increasing number of on-chip monitors are expected to be deployed for ensuring high reliability, high performance and low power. A dedicated interconnect subsystem such as MNoC provides an efficient, lightweight and programmable on-chip monitor data communication solution. Different types of on-chip monitors for measuring temperature, critical path delay, processor error, processor performance among others can be seamlessly integrated using MNoC and used in several monitoring applications to achieve power, reliability and performance benefits.

In this thesis, the thermal, error and AVF monitors that are spread across up to 32 cores are integrated using MNoC. The information from these monitors are sampled in real-time by a central controller, MEP, and used for remedial applications such as shared memory error recovery, and reliability-and-thermal-aware DVFS for the multicore system.

Use of MNoC for shared memory recovery approach is shown to provide flexibility in terms of selective recovery of only affected processors. This approach is found to be highly scalable for up to 32 cores since it suffered minimal performance degradation (less than 4%) as the monitor data communication delays increased.

Looking further the remedial system was expanded to include AVF and thermal monitors in addition to the error monitors, for DVFS applications. Previous work has

suggested a considerable impact of voltage and frequency variations on the raw SER. AVF has indicated the presence of an inherent architectural masking that alleviates the effective SER to a large extent. For a given target effective SER, amount of redundancy to be enabled in the system can be smartly decided based on the run time AVF of the structures in the system, while being also sensitive to the raw SER fluctuations. In this collaborative monitoring approach, a 6% reduction in power is achieved versus always-active redundancy while a stable multicore effective SER is maintained.

BIBLIOGRAPHY

[1] S. Madduri, R. Vadlamani, W. Burleson, R. Tessier, "A Monitor Interconnect and Support Subsystem for Multicore Processors," In the Proceedings of the IEEE/ACM Design Automation and Test in Europe Conference, Nice France, April 2009.

[2] M. Prvulovic, Z. Zhang, J. Torrellas, "ReVive: CostEffective Architectural Support for Rollback Recovery in SharedMemory Multiprocessors," In Proceedings 29th Annual International Symposium, May 2002, pp 111 – 122.

[3] Kun-Lung Wu, W. K. Fuchs, J. H. Patel, "Error Recovery in Shared Memory Multiprocessors Using Private Caches," IEEE Transactions on Parallel and Distributed Systems, Volume 1, Issue 2, April 1990, pp 231 – 240.

[4] S. Reinhardt, S. Mukherjee, "Transient Fault Detection via Simultaneous Multithreading," In Proceedings of the International Symposium on Computer Architecture (ISCA), pages 25–36, June 2000.

[5] Fault-tolerant computer system design. Dhiraj K. Pradhan, Pages: 135 - 138 1996 ISBN:0-13-057887-8

[6] M. J. Iacoponi, "Hardware assisted real-time rollback in the advanced fault-tolerant data processor," Digital Avionics Systems Conference, 1991. Proceedings., IEEE/AIAA 10th , vol., no., pp.269-274, 14-17 Oct 1991.

[7] A. Maheshwari, W. Burleson, R. Tessier, "Trading Off Transient Fault-tolerance and Power Consumption in Deep Submicron VLSI Circuits," In IEEE Transactions on VLSI Systems, vol 12, no. 3, March 2004, pp. 299-311.

[8] S.S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, T.Austin, "A Systematic Methodology to Compute the Architectural Vulnerability Factors of a High-Performance Microprocessor," in Proc. 36th Ann. Int'l Symp. Microarchitecture (MICRO-36), IEEE CS Press, 2003.

[9] K.R. Walcott, G. Humphreys, S. Gurumurthi, "Dynamic Prediction of Architectural Vulnerability From Microarchitectural State," Proceedings of the International Symposium on Computer Architecture, pages 516-527, June 2007.

[10] L. Xiaodong, S. V. Adve, P. Bose, J. A. Rivers, "Online Estimation of Architectural Vulnerability Factor for Soft Errors," Computer Architecture, 2008. ISCA '08. 35th International Symposium on , vol., no., pp.341-352, 21-25 June 2008.

[11] A. Biswas, et al., “Quantized AVF: A Means of Capturing Vulnerability Variations over Small Windows of Time”, IEEE Workshop on Silicon Errors in Logic - System Effects, March 2009.

[12] D. Zhu, et al., “The effects of energy management on reliability in real-time embedded systems”, in the Proc. of the IEEE/ACM International Conference on Computer Aided Design, 2004.

[13] Z. Baoxian, H. Aydin, and D. Zhu, “Reliability-aware Dynamic voltage scaling for energy-constrained real-time embedded systems”, in the Proc. of the IEEE Conference on Computer Design, pp. 633-639, Oct. 2008.

[14] International Technology Roadmap for Semiconductors. http://www.itrs.net/

[15] J. Srinivasan, S. V. Adve, P. Bose, J. A. Rivers, “The Impact of Technology Scaling on Lifetime Reliability,” Proceedings of International Conference on Dependable Systems and Networks (DSN '04) June 2004.

[16] S. Borkar, “Challenges in Reliable System Design in the Presence of Transistor Variability and Degradation,” IEEE Micro, vol. 25, no. 6, Nov.-Dec. 2005, pp. 10-16.

[17] P. Shivakumar et al. “Modeling the effect of technology trends on the soft error rate of combinational logic,” In Proceedings of the International Conference on Dependable Systems and Networks, June 2002, 389–398.

[18] S. Shyam, K. Constantinides, S. Phadke, V. Bertacco, T. Austin, “Ultra Low-Cost Defect Protection for Microprocessor Pipelines,” International Conference on Architectural Support for Programming Languages and Operating Systems, October 2006.

[19] J. Smolens, B. Gold, J. Kim, B. Falsafi, J. Hoe, A. Nowatzyk, “Fingerprinting: Bounding Soft-Error Detection Latency and Bandwidth,” In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pages 224–234, October 2004.

[20] T.J Slegel et al. “IBM’s S/390 G5 Microprocessor Design.” IEEE Micro, pp 12-23, March/April 1999.

[21] Y. Tamir, M. Tremblay, D. A. Rennels, “The Implementation and Application of Micro Rollback in Fault-Tolerant VLSI Systems,” Fault-Tolerant Computing, 1988. FTCS-18, Digest of Papers., Eighteenth International Symposium, June 1988, pp 234 – 239.

[22] R. Teodorescu, J. Nakano, J. Torrellas, “SWICH: A Prototype for Efficient Cache-Level Checkpointing and Rollback,” IEEE Micro, Sept.-Oct. 2006. Volume: 26, Issue: 5, pp. 28-40.

[23] D. J. Sorin, M. M. K. Martin, M. D. Hill, D. A. Wood, "SafetyNet: improving the availability of shared memory multiprocessors with global checkpoint/recovery," 29th Annual International Symposium on Computer Architecture, Anchorage, AK. May 25-29, 2002.

[24] C. Weaver, J. Emer, S. S. Mukherjee, S. K. Reinhardt, "Techniques to reduce the soft error rate of a high-performance microprocessor," Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on, vol., no., pp. 264-275, 19-23 June 2004.

[25] S. S. Mukherjee, C. T. Weaver, J. Emer, S. K. Reinhardt, T. Austin, "Measuring architectural vulnerability factors," Micro, IEEE , vol.23, no.6, pp. 70-75, Nov.-Dec. 2003.

[26] N. Wang , M. Fertig, S. Patel, "Y-Branches: When You Come to a Fork in the Road, Take It," Proc. 12th Int'l Conf. Parallel Architectures and Compilation Techniques (PACT 03), IEEE CS Press, 2003, pp. 56-67.

[27] N. Soundararajan, et al., "Impact of DVFS on the architectural vulnerability of GALS architectures", in the Proc. of the Int'l Symposium on Low Power Electronics and Design, August 2008.

[28] T. Siddiqua and S. Gurumurthi, "Balancing Soft Error Coverage with Lifetime Reliability in Redundantly Multithreaded Processors", in the Proc. of International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems, Sept. 2009.

[29] S. Mukherjee, J. Emer and S. Reinhardt, "The soft error problem: an architectural perspective", in the Proc. of International Symposium on High-Performance Computer Architecture, pp. 243-247, 2005.

[30] A. Golander, S. Weiss and R. Ronen, "DDMR: Dynamic and Scalable Dual Modular Redundancy with Short Validation Intervals", in Computer Architecture Letters, vol. 7, issue 2, pp. 65-68, 2008.

[31] D. Ernst, et al., "Razor: circuit-level correction of timing errors for low-power operation", IEEE Micro, vol. 24, no. 6, pp.10-20, Nov.-Dec. 2004.

[32] S. Velusamy, W. Huang, J. Lach, M. Stan, and K. Skadron, "Monitoring Temperature in FPGA based SOCs" , In Proceedings of the International Conference on Computer Design, Oct 2005, San Jose, CA

[33] R. McGowen, C. A. Poirier, C. Bostak, J. Ignowski, M. Millican, W.H. Parks, S. Naffziger, "Power and Temperature Control on a 90nm Itanium Family Processor", IEEE Journal on Solid State circuits , vol. 41, no 1, Jan 2006 , pp. 229-237.

[34] K. A. Bowman, J. W. Tschanz, N. Sung Kim, J. C. Lee, C. B. Wilkerson, Shih-Lien L. Lu, T. Karnik, V. K. De, "Energy-Efficient and Metastability-Immune Timing-Error Detection and Instruction-Replay-Based Recovery Circuits for Dynamic-Variation Tolerance," ISSCC 2008.

[35] M. Saen, K. Osada, S. Misaka, T. Yamada, Y. Tsujimoto, Y. Kondoh,T. Kamei, Y. Yoshida, E. Nagahama, Y. Nitta,T. Ito, T. Kameyama, N. Irie, "Embedded SoC Resource Manager to Control Temperature and Data Bandwidth," ISSCC 2007.

[36] M.S. Floyd, S. Ghiasi, T.W Keller, K. Rajamani, F.L. Rawson, J. C. Rubio, M. S. Ware, "System Power Management Support in the IBM Power6 Microprocessor," IBM Journal of Research and Development, vol. 51, no 6, Nov 2007.

[37] S. Das, D. Roberts, L. Seokwoo, S. Pant, D. Blaauw, T. Austin, K. Flautner, T. Mudge, "A self-tuning DVS processor using delay-error detection and correction," Solid-State Circuits, IEEE Journal of , vol.41, no.4, pp. 792-804, April 2006.

[38] D. Blaauw, S. Kalaiselvan, K. Lai, Wei-Hsiang Ma, S. Pant, C. Tokunaga, S. Das, D. Bull, "RazorII: In-Situ Error Detection and Correction for PVT and SER tolerance," IEEE International Solid-State Circuits Conference (ISSCC), February 2008.

[39] P. Hazucha, C. Svensson, "Impact of CMOS technology scaling on the atmospheric neutron soft error rate", In: Proc. of IEEE Transactions on Nuclear Science, 2000, pp. 2586-2594.

[40] B. Sprunt, "Pentium 4 Performance-Monitoring Features", IEEE Micro, vol. 22, no. 4, pp. 72-82, 2002.

[41] X. Fan, C. Ellis, and A. Lebeck, "The synergy between power-aware memory systems and processor voltage", in the Proc. of the Workshop on Power-Aware Computing Systems, 2003.

[42] K. Skadron, et al., "Temperature-aware microarchitecture: Modeling and implementation", in ACM Transactions on Architecture and Code Optimization, vol. 1 no. 1, pp. 94-125, Mar. 2004.

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

[44] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," in the Proc. of the International Symposium on Computer Architecture, June 2000.

[45] P. Shivakumar and N. Jouppi, “CACTI 3.0: An integrated cache timing, power and area model,” in Technical Report 2001/2, Compaq Computer Corporation, August 2001.

[46] G. Link and N. Vijaykrishnan, “Thermal trends in emerging technologies,” Proc. of the Int’l Symposium on Quality Electronic Design, Mar. 2006.

[47] A. Biswas, P. Racunas, R. Cheveresan, J. Emer, S. S. Mukherjee, R. Rangan, “Computing architectural vulnerability factors for address-based structures,” Computer Architecture, 2005. ISCA '05. Proceedings. 32nd International Symposium on , vol., no., pp. 532-543, 4-8 June 2005.

[48] A. Drake, R. Senger, H. Deogun, G. Carpenter, S. Ghiasi, T. Nguyen, N. James, M. Floyd, V. Pokala, “ A Distributed Critical-Path Timing Monitor for a 65nm High-Performance Microprocessor,” In Proceedings of the IEEE International Solid-State Circuits Conference , Feb 2007.

[49] B. Froba, C. Rothe, C. Kublbeck, “Evaluation of sensor calibration in a biometric person recognition framework based on sensor fusion,” Fourth IEEE International Conference on Automatic Face and Gesture Recognition, 2000, pp.512 – 517, 2000.

[50] B. T. Gold, J. C. Smolens, B. Falsafi, J. C. Hoe, “The Granularity of Soft-Error Containment in Shared Memory Multiprocessors,” 2006 Workshop on System Effects of Logic Soft Errors, April 2006.

[51] D. Truong, W. Cheng, T. Mohsenin, Z. Yu, T. Jacobson, G. Landge, M. Meeuwsen, C. Watnik, P. Mejia, A. Tran, J. Webb, E. Work, Z. Xiao, B. M. Baas, “A 167-processor 65 nm Computational Platform with Per-Processor Dynamic Supply Voltage and Dynamic Clock Frequency Scaling,” Symposium on VLSI Circuits, (VLSI '08), July 2008, C3.1.

[52] E. Boemo, S. Lopez-Buedo, “Thermal Monitoring on FPGAs using Ring-Oscillators,” In Proceedings of the Seventh International Workshop on Field Programmable Logic and Applications , Sep 1997, London, UK.

[53] E. Chi , A. M. Salem, R.I. Bahar, R. Weiss, “Combining Hardware and Software Monitoring for Improved Power and Performance Tuning,” In Proceedings of the Seventh Workshop on Interaction between Compilers and Computer Architectures , Anaheim, CA, Feb 2003.

[54] G. G. Yen, W. Feng, “Intelligent sensor validation by a hierarchical mixture of experts network,” Industrial Electronics Society, 2000. IECON 2000. 26th Annual Confjerence of the IEEE , vol.1, no., pp.155-160 vol.1, 2000.

[55] G. Yalcin, O. Ergin, “Using Tag-Match Comparators for Detecting Soft Errors,” IEEE Computer Architecture Letters, vol. 6, no. 2, pp. 53-56, Jul-Dec, 2007.

[56] I. Koren, C. M. Krishna, “Fault-Tolerant Systems”. Amsterdam; Boston : Elsevier/Morgan Kaufmann, 2007.

[57] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Palo Alto, CA, U.S.A.:Morgan Kaufmann, 1988.

[58] J. Smolens et al. , “Detecting emerging wearout faults,” In SELSE ’07: Procs. of the 3rd Workshop on Silicon Errors in Logic - System Effects, 2007.

[59] J. Blome, S. Feng, S. Gupta, S. Mahlke, “Self-calibrating Online Wearout Detection,” Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on , vol., no., pp.109-122, 1-5 Dec. 2007.

[60] L. Shang, L. Peh, N. K. Jha, "Dynamic voltage scaling with links for power optimization of interconnection networks," International Symposium on High-Performance Computer Architecture, pp. 91-102, Feb. 2003.

[61] M. A. Holtz, S. Narasimhan, S. Bhunia, “On-Die CMOS Voltage Droop Detection and Dynamic Compensation,” GLSVLSI’08, pp. 35-40, May 4–6, 2008, Orlando, Florida, USA.

[62] N. Bartzoudis, K. McDonald-Maier, “An adaptive processing node architecture for validating sensors reliability in a wind farm,” Bio-inspired, Learning, and Intelligent Systems for Security, 2007. BLISS 2007. ECSIS Symposium on , vol., no., pp.83-86, 9-10 Aug. 2007.

[63] S. Madduri, W. Burleson, R. Tessier, “A Simulation Study of Monitor Network-on-Chip Protocols and their Interface with the Monitor Executive Processor Used for Control,” SRC Report. Sept 2007.

[64] S. Mukherjee, M. Kontz, S. Reinhardt, “Detailed Design and Evaluation of Redundant Multithreading Alternatives,” In International Symposium on Computer Architecture (ISCA), pages 99–110, May 2002.

[65] T. Austin, D. Blaauw, T. Mudge, K. Flautner, “Making Typical Silicon Matter with Razor,” Computer, vol. 37, no. 3, pp. 57-65, Mar., 2004.

[66] V. Bychkovskiy, S. Megerian, D. Estrin, M. Potkonjak, “A Collaborative Approach to In-Place Sensor Calibration,” Center for Embedded Network Sensing, Technical Reports, pp. 59, 2003.

[67] Z. Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zyuban, H.Jacobson, P. Bose, “Microarchitectural Techniques for Power Gating of Execution Units,” International Symposium on Low Power Electronics and Design, 2004.

[68] H. Homayoun; T. H. Szymanski, "Reducing the Instruction Queue Leakage Power in Superscalar Processors," Electrical and Computer Engineering, 2006. CCECE '06. Canadian Conference on , vol., no., pp.1685-1689, May 2006.

[69] V. Stojanovic, R. I. Bahar, J. Dworak, R. Weiss, "A cost-effective implementation of an ECC-protected instruction queue for out-of-order microprocessors" In Proceedings of the 43rd Annual Design Automation Conference (San Francisco, CA, USA, July 24 - 28, 2006). DAC '06. ACM, New York, NY, 705-708.