IMPROVING THE RELIABILITY AND POWER-EFFICIENCY
CHARACTERISTICS OF EMERGING MANY-CORE
MULTIPROCESSORS

A Dissertation in
Computer Science and Engineering
by
Hui Zhao

© 2014 Hui Zhao

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

December 2014
The dissertation of Hui Zhao was reviewed and approved* by the following:

Mahmut Kandemir  
Professor of Computer Science and Engineering  
Dissertation Co-Adviser, Co-Chair of Committee

Mary Jane Irwin  
Professor of Computer Science and Engineering  
Dissertation Co-Adviser, Co-Chair of Committee

Padma Raghavan  
Professor of Computer Science and Engineering

Daniel Haworth  
Professor of Mechanical Engineering

Lee Coraor  
Associate Professor, Graduate Program Chair of Department of Computer Science and Engineering

*Signatures are on file in the Graduate School.
Abstract

In recent years, many-core multiprocessors have become the focus of attention in computer architecture design. Designers are permitted by Moore’s Law to integrate a large number of cores in a single chip. Processors containing close to a hundred of cores will appear in the near future. Using multiple cores on a single chip can significantly boost the performance of microprocessors. However, as the semiconductor industry has evolved into the deep sub-micron era, power consumption becomes the constraining factor to the further enhancement of the processor’s performance. Meanwhile, reliability has become an issue when various power management techniques are applied to reduce the power consumption. These have led to the need for designs that carefully balance the trade-offs of all design options in order to achieve high reliability and power-efficiency without harming performance. This dissertation is a step towards developing many-core multiprocessors that address reliability and power-efficiency issues.

In this thesis, we address two important multiprocessor characteristics - reliability and power-efficiency. Given that caches and interconnection networks in multicores are two structures playing important roles in storing and transmitting
program data, their reliability directly affects the correctness of program executions. We first provide a microarchitectural solution that is based on control theory to protect a multicore’s caches against transient errors. Transient errors are caused by particle strikes such as neurons or alpha particles. Caches are particularly susceptible to transient errors due to their large sizes. Our scheme takes two input parameters into account: performance Quality of Service (QoS) requirement and reliability Quality of Service requirement. The performance QoS indicates the minimum cache hit rate value acceptable, whereas the reliability QoS represents the desired reliability assurance. By balancing the partitioned cache spaces allocated to data and their replicas, our proposed scheme is able to provide both performance and reliability guarantees.

To ensure correct data transmission, on-chip networks usually employ error correction codes to protect the data. However, prior work has mostly employed fixed data retransmission schemes when an error is detected. We propose a flexible scheme that dynamically chooses the time for error checking and retransmission and takes advantage of both the end-to-end retransmission and the hop-by-hop retransmission in a many-core multiprocessor. Our scheme not only meets the NoC reliability requirements but also improves the power-efficiency.

Besides their reliability design requirements, caches and networks also need careful consideration with respect to their power characteristics. On-chip networks and caches are two major power consumers in many-core multiprocessors aside from the cores. To reduce the power and area costs of on-chip networks, schemes have been proposed to design networks with bufferless routers. However, bufferless routers only bring benefits when network utilization is low. We pro-
vide a solution with a heterogeneous NoC design that employs both buffered and bufferless routers in the same network. We explore the design space by evaluating different router placements in order to achieve optimal performance with maximum power-efficiency. In order to fully take advantage of the heterogeneous NoC architecture, we also design algorithms for application mapping and packet routing.

Through our evaluation of parallel programs, we observe that program performances exhibit different scalability characteristics with the number of cores and the size of the caches. Running programs with more cores or larger caches does not always result in better performance but incur larger power overheads. Based on this observation, we propose a dynamic scheme to allocate power to cores and caches. Our scheme first predicts the scalability of parallel programs. Then some cores or caches are selectively gated off in order to save power. By taking advantage of the program scalability, our scheme can achieve near optimal performance with the power available.
Table of Contents

List of Figures x
List of Tables xv
Acknowledgments xvi

Chapter 1
Introduction 1
1.1 Performance, Energy and Reliability: Closely Inter-related Features of Many-Core Multiprocessors 2
1.2 Improving Reliability of Many-Core Multiprocessors 5
1.3 Improving Energy-efficiency of Many-Core Multiprocessors 7
1.4 Organization 9

Chapter 2
Feedback Control Based Cache Reliability Enhancement 10
2.1 Introduction 11
2.2 Architectural Abstraction for Replication in Cache 13
2.3 QoS Metrics, Control Parameters, and Problem Definition 16
2.4 Control Architecture 19
   2.4.1 Background on Formal Feedback Control 19
   2.4.2 Our Adaptive Control Design 20
   2.4.3 Performance Adaptive Controller 21
   2.4.4 Reliability Adaptive Controllers 23
   2.4.5 Actuator 24
   2.4.6 Flow of Events 26
2.5 Experiments 26
Chapter 5
Exploiting Program Parallelism to Improve the Power-Efficiency of Many-Core Multiprocessors

5.1 Introduction
5.2 Motivation
5.2.1 On chip power breakdown
5.2.2 Target CMP architecture and power re-allocation schemes
5.3 TaPEr: Our Proposed Power Re-allocation Strategy
5.3.1 Measurement of performance
5.3.2 Predicting scalability
5.3.3 Static TaPEr vs. dynamic TaPEr
5.4 Experimental Evaluation
5.4.1 Experimental setup
5.4.2 Evaluation results
5.4.2.1 Case study I: different workloads under the same power budget
5.4.2.2 Case study II: same workload with different power budgets
5.4.2.3 Case study III: same workload with different program phases
5.4.2.4 Overall results across all workloads
5.4.2.5 Sensitivity analysis
5.5 Related Work
5.6 Conclusion

Chapter 6
Conclusion and Future Work

6.1 Contribution
6.2 Future Work
6.2.1 Exploiting Heterogeneity in Network-on-Chip Design
6.2.1.1 Providing QoS Assurance in Low-cost NoCs
6.2.1.2 Improving Reliability in Low-cost NoCs
6.2.2 Exploiting program parallelism to increase power-efficiency for multiprogrammed applications
### List of Figures

1.1  Total chip dynamic and static power trends based on ITRS projections [2]. ........................................... 3

1.2  Interrelationship between performance, power and reliability characteristics of many-core multiprocessors. ........ 5

2.1  Interrelationship between performance, power and reliability characteristics of many-core multiprocessors. ........ 11
2.2  Different multicore architectures with on-chip L2 caches. .............. 13
2.3  Different cases where an L1 cache may be shared by multiple applications. ........................................... 14
2.4  Partitioning the shared L2 cache space into a shared area for storing original blocks and a replication area divided across applications. In our work there are two control parameters (knobs). The effective cache space is the main parameter to control the performance QoS. On the other hand, the per-application cache space allocation in the replication area is the control parameter to control the reliability QoS values. ........................................... 18
2.5  Illustration of a simple feedback control loop for reliability management in a CMP. ........................................... 19
2.6  High level view of our control architecture for the proposed reliability enhancement when targeting a four core machine with a shared L2 cache. Regions R1 through R4 are the replication areas in the cache. Both the shared L2 cache size and the size of the replication area are controlled dynamically by the Performance Adaptive Controller (PAC) and Reliability Adaptive Controller (RAC) components, respectively, to achieve the specified performance and reliability QoS metrics. The actuator plays the role of an arbitrator which finds a feasible cache allocation from independently determined demands by the PAC and RACs. ........................................... 20
2.7 Hit rate values of the effective cache space under different sizes of the effective cache area when the replication area is partitioned "equally" among four applications in a workload. The configurations on the x-axis are the allocations for [effective: replica1: replica2: replica3: replica4], where effective is the cache ways allocated to the effective cache area and replica1 through replica 4 are the cache ways allocated to the four applications in the replication area.

2.8 RwR values with under different sizes of the replication area when it is partitioned equally among four applications in a workload. The configurations on the x-axis are the allocations for [effective: replica1: replica2: replica3: replica4], where effective is the cache ways allocated to the effective cache area and replica1 through replica 4 are the cache ways allocated to the four applications in the replication area.

2.9 Performance targets achieved by our scheme. The numbers in the x-axis indicate the target performance QoS values.

2.10 Replication targets achieved by our scheme. A uniform target RwR value of 0.6 is set for all applications in all workloads.

2.11 Modulations in effective cache space to track the target performance QoS (Mix 7).

2.12 Tracking the hit rate by modulating the effective cache space (Mix 7).

2.13 Variations in instructions per cycle (IPC) by modulating the effective cache space (Mix 7).

2.14 Partitioning of the replication area (Mix 7).

2.15 Tracking of RwR target for the four applications of Mix 7 under different sizes of the replication area when it is partitioned by our feedback controlled replication scheme.

2.16 Sensitivity to performance QoS target.

2.17 Sensitivity to reliability QoS targets. X-axis is RAC targets for all applications.

2.18 Sensitivity to execution intervals of RACs and the PAC (R,P) for Mix 7.

2.19 Performance of RACs when our scheme is used with a 16 KB L1 cache shared by four cores.

3.1 Interrelationship between performance, power and reliability characteristics of many-core multiprocessors.
3.2 A generic NoC router architecture with M PCs (Physical Channel) and N VCs (Virtual Channel)........................................... 39
3.3 Effect of load rate and error rate on performance.................. 39
3.4 Effect of load rate and error rate on power.......................... 40
3.5 Power and performance tradeoffs with ECC checked at different time. 43
3.6 A typical mesh NoC based multicore system.......................... 44
3.7 Hardware modification on ECC decoding.............................. 45
3.8 Hardware modification on ECC encoding.............................. 45
3.9 Modified flit header to support dynamic hops per ECC............. 46
3.10 A high level view of feedback control system....................... 48
3.11 Average packet delay (fixed load rate and error rate)............ 51
3.12 Hops per ECC (fixed load rate and error rate).................... 51
3.13 Savings on error control energy (fixed load rate and error rate) 52
3.14 Average packet delay (fixed error rate, changing load rate)...... 52
3.15 Hops per ECC (fixed error rate, changing load rate)............ 52
3.16 Savings on error control energy (changing load rate)............ 52
3.17 Performance (fixed load rate, changing error rate).............. 54
3.18 Hops per ECC (fixed load rate, changing error rate)............ 54
3.19 Savings on error control energy (fixed load rate, changing error rate) 54
3.20 Performance (multiple applications with overlapping paths).... 54
3.21 Hops per ECC (multiple applications with overlapping paths).... 55
3.22 Savings on error control energy (multi-applications with overlapping paths) 55
3.23 Average packet energy consumption using different ECC coding techniques.................. 55
3.24 Power overhead of error control using different ECC coding techniques........................................... 55
4.1 Interrelationship between performance, power and reliability characteristics of many-core multiprocessors.................. 60
4.2 Optimal and non-optimal deflection................................. 62
4.3 Different placement plans for buffered and bufferless routers on a proposed heterogeneous NoC.......................... 64
4.4 Normalized performance of different router placements............ 65
4.5 Normalized power of different router placements.................. 65
4.6 Performance of router placements with different buffer size........ 65
4.7 Power of router placements with different buffer size............. 65
4.8 Normalized savings in buffer area.................................... 66
4.9 NoC area savings.................................................. 66
4.10 BRAM mapping with grid placement. ........................................... 68
4.11 BRAM mapping with diagonal placement. ........................................ 68
4.12 BRAR routing with grid placement. .............................................. 70
4.13 BRAR routing with diagonal placement. ........................................... 70
4.14 Performance of BRAM and BRAR on a NoC of 64 nodes with diagonal placement. ......................................................... 72
4.15 Performance of BRAM and BRAR on a NoC of 64 nodes with grid placement. ......................................................... 72
4.16 Performance of BRAM and BRAR on a NoC of 100 nodes with diagonal placement. ......................................................... 73
4.17 Performance of BRAM and BRAR on a NoC of 100 nodes with grid placement. ......................................................... 73

5.1 Interrelationship between performance, power and reliability characteristics of many-core multiprocessors. .............................................. 76
5.2 Power breakdown between cores and caches (L2) under different technology nodes. The size of L2 caches ranges from 64 KB up to 1MB. ......................................................... 79
5.3 The CMP architecture and power re-allocation scenarios due to power emergency investigated in this work. Cores or caches that are gated off are represented by dark regions. Scenario (a) represents a baseline case where the CMP is running with full power budget. Scenarios (b)-(d) illustrate possible power re-allocation methods that satisfy a same reduced power budget. ......................................................... 80
5.4 Characterization of parallel applications showing different scalabilities. The execution time of each application is normalized with respect to that of the full configuration (48 cores and 1MB private L2 cache). Each application is run with 48 threads. ......................... 81
5.5 Plots of real and predicted performance for apsi. ........................... 85
5.6 Example of workloads with different phases. ................................. 87
5.7 Evaluation of schemes on workloads with different scalabilities. .......... 90
5.8 Evaluation of schemes with changing power budgets using apsi where power budget changes between 90% and 60% of the full configuration. ......................................................... 91
5.9 Evaluation of schemes with dynamically changing phases using canneal. ......................................................... 92
5.10 Comparison of normalized performance of configuration schemes when the power budget is reduced to 80%. ......................................................... 93
5.11 Comparison of normalized performance of configuration schemes when the power budget is reduced to 60%.

5.12 Comparison of normalized performance of configuration schemes when the power budget is reduced to 80%.

5.13 Comparison of normalized performance of configuration schemes when the power budget is reduced to 60%.

6.1 Heterogeneous NoC architecture with buffered routers providing QoS support.

6.2 Packet retransmission with (a) bufferless NoC or (b) heterogeneous NoC.

6.3 Allocate core/cache power to multiple parallel applications based on the programs’ scalability.
List of Tables

2.1 Baseline configuration. ........................................... 26
2.2 Various mixes of the SPEC CPU2006 applications considered for our multiprogrammed workload. The third, fourth, and fifth columns show, respectively, the average L1 miss rate, L2 miss rate, and sum instructions per cycle of the different workloads when using the baseline configuration described in Table 5.1 without our scheme. 27

3.1 Control system components and corresponding parameters in our design. ........................................... 47
3.2 NoC Network Configuration. ....................................... 51
3.3 NoC Component Power Consumptions(mW) ........................................... 51

4.1 Baseline configuration. ........................................... 71
4.2 Applications formed with a pair of OpenMP multi-thread benchmarks. ........................................... 71
4.3 Mix of applications mapped to a 8x8 NoC. ........................................... 71

5.1 Baseline CMP configuration. ........................................... 88
Acknowledgments

First of all, I would like to express my sincere gratitude to my advisors: Dr. Mahmut Kandemir and Dr. Mary Jane Irwin.

When I am stuck with my projects, they are always ready to help me and direct me to move forward. Dr. Kandemir not only amazed me with his intellectual power but also his work ethics of never stopping challenging himself. Dr. Irwin continuously inspires the female students as a role model to pursue a career in science and technology. This dissertation would not have been possible without their continuous support, insightful advice and unsurpassed knowledge of my research field.

I also want to thank Dr. Padma Raghavan. My research experience starts with the involvement in the PxP discussion group together with her. I am also grateful to Dr. Daniel Haworth for serving in my thesis committee.

In addition, I would like to thank the former and current members at the Microsystems Design Laboratory for their support and inspiring discussions. Although I do not have enough space to list each of them, I am particularly thankful to Professor Yuan Xie, Professor Vijay Narayanan, and the MDL students who have helped me.

Finally, I would like to thank my family. I am deeply indebted to my parents. They have always stood by me and helped me with selfless sacrifice. I could not finish this dissertation without their support. I dedicate this dissertation to my beloved parents. I also dedicate this thesis to my lovely children, Michelle and Angela. They made sacrifice to help me finish my study and bring joy into my life everyday.
To the memory of my mother Shuqin Zhao...
Chapter 1

Introduction

According to Moore’s Law, the number of transistors on the integrated circuits doubles approximately every two years. During the last decades, transistor count on a single chip has grown exponentially as modern fabrication technologies advance into deep submicron era. Chips with billions of transistors have already come into exist: such as AMD’s Trinity, Intel’s Itanium and IBM’s POWER7. As of 2014, among the commercially available CPUs, Intel’s Xeon IvyBridge-EX has the highest transistor count which is 4.3 billion. Such enormous number of transistors cannot be fully taken advantage by the uniprocessors due to several reasons: Firstly, the performance of uniprocessors is not scalable because only limited amount of parallelism can be extracted from the conventional super-scalar techniques. Secondly, clock speed cannot be increased without limit due to the power dissipation problems. Thirdly, simpler processors are more preferable due to the lower design and debugging costs.

As a result, designers resort to on-chip multiprocessors to integrate multiple simple processor cores into a single die. Evolved from the initial dual-core and quad-core processors, the multi-core processors are moving toward many-core processors. For example, Sun’s SPARC T5 has 16 cores and can run 128 concurrent threads in parallel. Intel’s next generation many-core product Xeon Phi can integrate up to 72 cores into one supercomputing chip. Emerging many-core multiprocessors can scale across generations of processing technologies to avoid the high
design and debugging costs. In addition, many-core multiprocessors can achieve higher performance by taking advantage of thread level parallelism. Because many processors are integrated into one die, this greatly reduces the distance between cores and leads to lower communication cost of parallel threads. While the many-core multiprocessors provide a promising solution to counter Moore’s Law, they also bring new challenges in designing such systems. Among them, reliability and power efficiency are two critical problems need to be solved.

1.1 Performance, Energy and Reliability: Closely Inter-related Features of Many-Core Multiprocessors

The last a few decades saw great improvements in microprocessor performances. Between the year 1978 and 1986, the average performance gain of microprocessors is 25% per year[1]. From year 1986 to year 2002, microprocessor performance increased by about 800 times, at a rate of 52% per year. After that, the speedup experienced a slowdown, but is still at 22% per year. Several factors contributed to such huge speedups. First of all, there are more transistors integrated into one chip. With the advance of fabrication technology, transistors shrink significantly in sizes. For example, the gate length of MOSFETs shrank from 10um in 1970 to 14 nm in 2014. Smaller sizes increase the density of transistors on a die. In addition, the die size becomes larger by time. As a result, large amount of transistors are available to perform more functionality, which in turn leads to increased performances. The second factor is the increased clock speed. Higher clocked frequency allows the program to run faster. The early VAX-11/780 processors can only run at frequencies of 5 MHz. In comparison, the recent Intel Xeon processors can reach the frequency of 3 GHz. The exploration of parallelism of programs is the third contributing factor. Many complicated techniques are applied to exploit program parallelism at various levels: such as instruction level parallelism, thread level parallelism and memory level parallelism. Such techniques allow programs to fully take advantage of the abundant transistors available and can greatly reduce the execution time.
However, in recent years, the trend of performance speedup got slowed down because it is impossible to continuously run at higher frequencies without hitting energy/power limits (power wall).

\[
\text{Energy} = (\text{Power} \times \text{Time}) = (\text{Dynamic Power} + \text{Static Power}) \times \text{Time}. \tag{1.1}
\]

\[
\text{Dynamic Power} \propto (\text{Activity} \times \text{Capacitance} \times \text{Voltage}^2 \times \text{Frequency}). \tag{1.2}
\]

The total power consumption consists of dynamic power and static power (leakage power). Dynamic power is directly related to the running frequency. Higher frequency leads to more dynamic power consumption. Traditional microprocessor designs focus on reducing the dynamic power consumptions. However, with the shrinking transistor sizes, the static power consumption also rises significantly and is projected to surpass the dynamic power consumption[2]. Static power is a function of number of transistors, leakage current, and supply voltage. Figure 1.1 shows the chip dynamic and static power consumption trends based on the statistics from International Technology Roadmap for Semiconductors (ITRS). As shown in this figure, with the continuously shrinking gate length, the static power grows at a faster speed than the dynamic power and it has already surpassed the dynamic power consumptions. As the sum of these two types of power consump-
tions, the rise of total power is prohibitive. Today, the total power consumptions of high-performance processors already exceed 100 W. Additionally, high power density causes thermal issues by heating up the processors quickly. As a result, the power consumptions of many-core multiprocessors greatly constraints performance improvements.

Power consumptions not only limit the performance improvements of many-core multiprocessors, but also directly affect another important characteristic: reliability. Circuit failures include hard failures and soft failures. Hard failure leads to complete loss of functionality of a device. Typically the hard failures include oxide breakdown and electromigration. They are either related with leakage current or electrical current. Soft failures are not permanent. They can cause devices to have timing errors or behave incorrectly. Both types of failures are related to the threshold voltage. To achieve best performance under certain power constraints, various power management techniques have been applied to many-core multiprocessors such as dynamic voltage and frequency scaling (DVFS), low-power modes and clock gating. However, these techniques can reduce the reliability and make the chip more error prone. For example, using techniques such as DVFS and low-power modes, power consumption can be reduced by tuning down just the supply voltage alone or both the supply voltage and frequency together. It is already shown that decreasing the supply voltage or frequency increases the Soft Error Rate (SER) of circuits [3, 4]. Clock gating is another technique to save dynamic power by turning off the clock of a particular region on the chip. Because extra physical gates need to be inserted to the clock paths in the clock gating techniques, clock skews and races could be introduced and lead to timing violations. As a result, the chip becomes less reliable. Under limited power budgets, there is also a tradeoff between performance and reliability. Designers need to balance the power budgets available to pursue better performance or higher reliability. Figure 1.2 shows the inter-relationship among power, performance and reliability of many-core processors. As we can see, higher power boosts the performance and can lead to lower error rates. Lower power limits the performance gains and can bring higher error rates. Thus, for many-core multiprocessor designs, power reduction is only half the challenge. The ultimate goal is to achieve high performances and maintain certain level of reliability while consuming less power.
1.2 Improving Reliability of Many-Core Multiprocessors

There have been various techniques proposed to increase the reliability without increasing the performance and power overhead. These techniques target on the major on-chip components respectively, such as CPUs, caches and interconnection networks.

Traditionally, CPU reliability problems are addressed through dual-module or triple-modular redundancy. However, the costs of such solutions are considered too high for current many-core multiprocessors which pursue architectures that are both economic and reliable. [8, 9] designed checkpointing schemes to ensure the functional correctness of the CPUs which periodically create checkpoints during program executions. Whenever an error state is detected, the processors can roll back to a checkpointed correct state and resume execution. [5] proposed an online testing solution to increase the CPU’s reliability. They first identify components that are more frequently used by the software applications and then test these components more thoroughly. Thus this approach can achieve high error coverage with less performance overhead. To increase the CPU’s tolerance of permanent faults, [6, 7] proposed architectures that are built with redundant hardware components.
Through runtime detection of failed components, the redundant components are employed to carry on the execution in order to avoid errors.

Another type of important components that need to be protected are on-chip caches. Caches are commonly built with SRAM cells that are vulnerable from single event upset (SEU) faults, which are caused by energetic particles such as neutron and alpha particles. In recent years, the on-chip cache sizes are continuously growing. This increases the chances of cache cells being hit by energetic particles. In addition, to save power consumptions, aggressive power scaling technologies are applied to caches which increased the cache error rates. Most commonly used techniques to provide cache reliability is through Error Correction Codes (ECC). Using these techniques, each block of cache data is associated with an ECC code. Whenever a block of cache data is written into the caches, the ECC encoding procedure is employed to generate the corresponding ECC codes. Then the ECC codes are stored together with the cache data. When a cache block is accessed later by a read operation, ECC decoding procedure is called and the error codes can tell if the data is corrupted or not. Based on the accuracy and complexity of design requirements, there are several types of ECC codes to choose. The simplest but least powerful ECC codes are parity codes which are only capable for error detection. More complicated ECC codes such as Hamming[10] or Reed Solomon (RS)[11] codes can not only find error but also have error correction capabilities. Other than employing ECC techniques, there have been several methods proposed to provide cache side reliability. Zhang et al [70] proposed a scheme, called In-Cache Replication (ICR), which utilizes the existing cache space to hold the replica of a cache block. This mechanism can be used either with ECC or with parity based schemes. PADded Cache [53] is proposed to increase cache reliability through a special Programmable Address Decoder (PAD). Faulty blocks are disabled and to data references are re-mapped to healthy blocks.

Errors occurring in NoCs can be categorized into permanent errors (hard errors) and transient errors (e.g. soft errors). Permanent errors are mostly caused by accelerated aging effects such as electromigration and manufacturing limitations. Transient errors, on the other hand, are mainly caused by aggressive shrinking of feature sizes of technology nodes. As devices shrink to the nanometer scale, the decrease in power supply voltage and device Vt leads to unreliable wires that
are increasingly susceptible to all kinds of noises. Transient errors in an on-chip interconnect can be caused by many factors including crosstalk, electromigration, electro-magnetic interference, alpha particles and cosmic radiations. It is critical for NoC designers to provide a certain level of resilience against such errors. There have been extensive studies in this area. For example, error detection and correction codes have been employed to protect against transient errors (e.g., parity, SECDED and FEC). Retransmission based schemes have also been proposed for error recovery. However, reliability is typically achieved at the expense of sacrificed performance and power. In particular, ECC CODECs and buffers used for error protection contribute to the power overhead. The power used for error correction can reach up to 20% of the overall network power in the current technology \cite{54}. Performance is also compromised because extra cycles are needed for error checking and correction. Therefore, it is critical to fully explore the design space to build a reliability scheme with low performance and power overhead. In order to detect errors and correct them on the earliest occurrence, Kim et al \cite{55} presented a flit based Hop-by-Hop retransmission scheme to handle uncorrectable errors. In this scheme, on every hop along the path, data needs to be checked by ECCs and written to retransmission buffers. Yu et al \cite{56} proposed an adaptive error control method for NoCs running in a variable noise environment. Nodes in the NoC have multiple level error control abilities. ECC schemes of different correction abilities are selected based on error rates. Based on the observations that some applications have inherent error tolerance, Yanamandra et al \cite{57} proposed a scheme to exploit the AVF factors in order to save power while ensuring certain reliability levels. Since not all transient faults result in errors in program outputs, error correction mechanisms are selectively turned off in their scheme to save power.

1.3 Improving Energy-efficiency of Many-Core Multiprocessors

DVFS is the most commonly used technique to manage CPU power. Today, various commercial-available processors provide users options to scale down voltage and frequency to reduce CPU power. DVFS techniques exploit the program CPU slack
time incurred by memory accesses or I/O operations. By scaling down the processor speed during these times, DVFS can effectively reduce the power consumption without hurting the performance. Clock throttling is another method to save CPU power consumptions. A real world example is Intel’s Pentium M processors that provide both these two types of techniques to manage CPU power. Besides the hardware based techniques, [12] proposed to coordinate the power management through the OS. [13] introduced a technique to collaboratively employ the OS and compilers together to reduce the CPU power. They used the compiler to annotate an application’s source code with hints about the application’s temporal behavior. Then the OS varies the processor frequency and voltage based on this temporal information so as to save power.

Besides CPUs, a sizeable portion of the total processor power is consumed by caches. Nowadays, on-chip cache sizes are continuously growing in order to enhance the processor’s performance. It is common to find two or three levels of caches on many-core multiprocessors. The dominant factor contributing to total cache power is the static power or the leakage power. Due to the memory access pattern of programs, only a subset of all of the cache blocks are active at one time. Based on this observation, several techniques were proposed to put the inactive cache blocks into low-power mode in order to save static power.[15, 16, 17, 140]. Other techniques resort to emerging non-volatile memory technologies instead of the traditional SRAM cells to counter the leakage power problem. Contrary to the traditional SRAMs, the non-volatile memories can still keep their data without power supply, which makes them attractive options to design power-efficient caches. [18] proposed to combine MRAM, PRAM with SRAM to build a hybrid cache architecture. Such caches can reduce the power consumptions while maintaining similar performance as compared to SRAM caches. [19] explores to use another type of non-volatile RAM(STT-RAM) to build on-chip caches in order to balance the power and performance trade-off.

On the network side, with the increasing demand for network bandwidth, the power consumed by the interconnection network becomes substantial [20]. This makes power one of the first-order constraints for NoC designers. It has been proposed to selectively shut off network components in order to reduce power consumption. Network links are dynamically turned on and off to minimize the
leakage power in [21, 22]. By controlling the local power consumption of the individual routers, the peak power consumption of the network can be constrained [23]. [24] applied DVFS to NoCs by partition Network-on-Chip into voltage-frequency islands. In their scheme, a given NoC is first partitioned into multiple voltage-frequency domains and the supply and threshold voltages are assigned to each domain such that the total power consumption is minimized under given performance constraints. Then an online feedback control mechanism is applied to dynamically adjust the operating voltage and frequency. In most NoC designs, buffers are employed for flow control and ensure reliability. However, buffers consume significant amount of power. However, buffers consume significant power: dynamic power for flits reading/writing and leakage power even when they are not occupied. An NoC design with bufferless routers are proposed in [107] in order to reduce the power and area overhead. In such architecture, deflection routing algorithms are employed to route packets when contentions occur. Such a bufferless architecture can significantly reduces the network energy consumption, while providing similar performance to that of buffered architectures when the network utilization is low.

1.4 Organization

The rest of the dissertation is organized as follows: Chapter 2 and Chapter 3 present two techniques designed to improve the reliability of emerging many-core multiprocessors using control theory. Chapter 2 discusses a scheme that selectively duplicate data blocks in order to improve the reliability of data caches. In Chapter 3, a packet retransmission scheme is presented that is designed to enhance the reliability of on-chip networks. Chapter 4 and 5 discuss schemes to design power-efficient many-core processors. Chapter 4 presents a heterogeneous NoC that employs both buffered and bufferless routers to improve performance and reduce the cost in terms of power and area. A technique called TaPEnEr is discussed in Chapter 5, that dynamically configures the number of running cores and sizes of caches in order to achieve better performance under certain power constraints. At last, Chapter 6 summarizes and concludes the dissertation.
Due to increasingly problematic effects of clock frequency on power consumption and heat generation, there is a shift in chip manufacturing from complex single core machines to simple multicore architectures. While this move helps with power and temperature related issues and holds the complexity of a single core somewhat static over time, it also brings its own set of problems. First, an effective use of these architectures requires parallelizing single-threaded applications. Second, increasing core counts and limited off-chip bandwidth can result in pressure on communication bandwidth and memory accesses, respectively. Third, it is not clear how system software should be structured/redesigned for these architectures. Despite these challenges, the chip manufacturers such as IBM, AMD, Sun, and Intel already have multicore products in the market [38, 34, 41, 51, 46], and one can expect these emerging architectures to be the building blocks of any future computer system from smart phones to laptops to desktops to supercomputers. It has been projected that future multicores will have several interesting characteristics, as pointed out in [30]. One of these is the large number and variety of on-chip resources including processor cores and shared cache components.

Focusing on data reliability, we propose a control theory centric approach designed to improve transient error resilience in shared caches of emerging multicores while satisfying performance goals. The proposed scheme takes, as input, two quality of service (QoS) specifications: performance QoS and reliability QoS. The first
of these indicates the minimum workload-wide cache (L2) hit rate value acceptable, whereas the second one captures the reliability bound on an application basis, with the help of a metric called the Reads-with-Replica (RwR). We present an extensive experimental evaluation of the proposed scheme on various workloads formed using the applications from the SPEC2006 benchmark suite. The proposed scheme is able to satisfy, in most of the tested cases, both performance and reliability QoS, by successfully modulating the total size of the data replication area and partitioning of this area among the co-runner applications. The collected results also show that our scheme achieves similar improvements under different values of the major simulation parameters. This work targets on improving many-core multiprocessor design with the shaded part as shown in Figure 2.1.

Figure 2.1: Interrelationship between performance, power and reliability characteristics of many-core multiprocessors.

2.1 Introduction

Continuously shrinking process technologies and extensive use of low-power operating modes increase transient error probabilities in computer chips [29]. As future
multicores promise even larger transistor counts and more sophisticated power management modes [30], one can expect transient errors to be even more problematic in the next generation multicores. What makes this problem more interesting and challenging is the fact that many current multicores host multiprogrammed workloads in which multiple, independent applications execute at the same time in the same multicore, sharing available on-chip resources such as caches. Each of these applications may have different sensitivities regarding transient errors and addressing these sensitivities at the same time can be very challenging.

One of the shared resources in multicores that have demanded extensive attention lately is shared cache space. Many commercial multicores come with large (L2 or L3) on-chip caches that can be shared across simultaneously executing applications in a workload. If each of these applications requires a certain level of resilience against transient errors in cache accesses, one needs to develop suitable strategies to satisfy these reliability requirements while also satisfying performance requirements of the workload. For example, an OS interrupt handler requires both high reliability and quick response. However, other applications handling financial transactions require high reliability but can tolerate some slack in performance. On the other hand, a background batch application with reliability provided by the software (in terms of re-execution if execution is determined incorrect) does not require high levels of performance/reliability. Focusing on this performance-aware, application-sensitive data reliability problem, we make the following contributions in this work:

- We present a control theory centric approach designed to address the problem not addressed by ECC based techniques: Provide different levels of reliability for applications that share a given on-chip cache space. Our proposed approach improves transient error resilience in shared caches while satisfying the specified performance goals. The proposed scheme takes, as input, two quality of service (QoS) specifications: performance QoS and reliability QoS. The first of these indicates the minimum workload-wide cache (L2) hit rate value acceptable, whereas the second one captures the reliability bound on an application basis, with the help of a metric called the Reads-with-Replica (RwR).
• We present an extensive experimental evaluation of the proposed scheme on various workloads formed using the applications from the SPEC2006 benchmark suite [33]. In our experiments, we also test our scheme under different QoS values and measure the sensitivity of our approach to other design parameters.

Our experimental evaluations indicate that the proposed scheme is able to satisfy, in most of the tested cases, both performance and reliability QoS by successfully modulating the total size of the replication area and partitioning of this area among the applications. The collected results also show that our scheme generates good results under different values of the major simulation parameters.

2.2 Architectural Abstraction for Replication in Cache

Figure 2.2 shows four different eight-core multicore architectures, each with a different on-chip cache hierarchy. (a) represents a pure private, two-layer on-chip cache hierarchy. Main advantages of this architecture include performance iso-
Figure 2.3: Different cases where an L1 cache may be shared by multiple applications.

lation, absence of inter-core cache contention, and fast and power-efficient data accesses, whereas its main drawback is data replication across different L2 caches, which tends to reduce the effective cache capacity. The shared cache architecture illustrated in (b) does not replicate data in the L2 space; however, destructive interactions among data accesses coming from different cores can cause performance problems. (c) depicts a design point between pure private and pure shared L2 caches. In this hybrid architecture, each core has a private L1 and each pair of cores share an L2 cache. Finally, (d) shows an eight core processor with a three level hierarchical cache structure. Each core is connected to an L1 cache as before and each pair of L1 caches is connected to an L2 cache. Further, in the next level, each pair of L2 caches is connected to an L3 cache. Most current multicore architectures have some sort of shared on-chip cache space (e.g., Intel Harpertown and Dunnington [31] correspond to Figures 2.2(a) and (d), respectively). While in this work we mainly focus on the architecture shown in Figure 2.2(b), our approach can be used with other shared cache based architectures as well. In addition, while our focus is on the shared L2 cache, our approach can be applied to L1 caches as well, when either (i) multiple hardware threads on a core share an L1, as in the case of Sun UltraSPARC T1 [47], or (ii) multiple applications are mapped to the same core. These two cases are depicted in Figure 2.3. We present the results when targeting shared L1 caches, later in Section 2.5.4.

Previously-proposed error checking schemes for caches mostly employ either
parity to provide single error detection or ECC to provide Single Error Correction, Double Error Detection (SECDED). There are several problems that arise in practice, when using such schemes for transient errors. First, many existing techniques cannot provide differential levels of protection for different applications. For applications that do not need high level of protections, this leads to unnecessary power and performance costs. On the other hand, for applications that need very high level of protection, techniques such as Double Error Correction, Triple Error Detection (DECTED) cannot be applied since it will need to be applied to every cell, and this leads to prohibitive overheads. Second, a 12.5% extra overhead is incurred in storing an 8 bit SECDED for a 64-bit data block. Third, while simple schemes such as parity may be able to meet high performance requirement, ECC based schemes may hurt performance especially with high-end processors that are clocked at high frequencies. The additional power required for maintaining informational redundancy and verifying data integrity is high in SECDED based mechanisms. Transient errors do not occur with very high frequencies, and it is important that one does not pay a performance penalty or power cost in the common case when these errors do not occur.

Zhang et al [70] proposed a scheme, called In-Cache Replication (ICR), which utilizes the existing cache space to hold the replica of a cache block. This mechanism can be used either with ECC or with parity based schemes. In either case, the scheme tries to maintain a replica of a given block in the cache space. Both the original and the replica are protected by using the same scheme (parity or ECC). Under this scheme, we first access the original copy of the block, and if an error is detected, we access the replica. When this scheme is used together with ECC, one can get better reliability than with a scheme that uses ECC uniformly for all cache lines. When used with parity on the other hand, it can provide higher detection and recovery capabilities without degrading performance or power consumption significantly. In this work, targeting shared on-chip caches, we adopt a variant of ICR. Specifically, when a data block is brought into the cache, we store a replica of it in a reserved area of the cache (referred to as the Replication Area in this work). Later, when the original block is accessed and we determine a problem with the block (e.g., wrong parity if the blocks are protected using parity), we access the replica (stored in the replication area). Since chances for both the origi-
inal copy and the replica to get affected by transient errors are very rare, through this strategy, one can achieve low-cost resilience against transient errors. Consider for example, the case when the replication based scheme is used with parity. The replica can be used to correct errors when the parity detects presence of an error in the original data block. In this way, we can correct single bit errors. Note that, if desired, one can even access both original copy and its replica at the same time and perform bit-by-bit comparison (in background) thereby providing better resilience over conventional ECC. In any case, having a replica for a data block increases error correction capabilities. While one may want to have replicas for all data blocks in the cache, in practice this is not possible as it can hurt performance (under a fixed on-chip cache space). It is also important to note that, in a multiprogrammed workload, each application can have different reliability requirements. For example, two applications can have significantly different working set sizes, and consequently, require different amounts of replication. In addition, there is notable difference between ICR and our scheme: ICR is proposed to protect L1 caches where ECC is not applicable due to performance overheads. Our scheme targets on shared caches (usually L2/L3) to provide differentiated protection levels for applications, which ECC can not provide.

2.3 QoS Metrics, Control Parameters, and Problem Definition

We consider two types of QoS metrics in this work: performance metric and reliability metric. Our performance metric is workload-wide and captures the behavior of all the applications in the workload combined. Note that, one can potentially use different metrics for that purpose; the one employed in this work is the shared cache hit rate. While other metrics like instructions per cycle (IPC) could have been used, we found that IPC is influenced by many other factors other than cache miss rate alone and cache miss rate can be controlled only by tuning the cache allocation. Therefore, in our feedback control based approach, we use hit rate as the target metric. However, we also report the IPC numbers where appropriate.

Our reliability metric on the other hand is Reads-with-Replica (RwR) which
indicates the fraction of cache reads where the accessed data block also has its replica in the cache space. Clearly, we want the value of our RwR metric to be as high as possible.\(^1\) Note however that, as explained earlier, a large number of replicas can hurt performance since original blocks and their replicas share the same on-chip cache space. As a result, it is simply not possible to have a replica ready for every data block.

In our architecture, we reserve a fraction of the total cache space as the Replication Area. The remaining area is referred to as the Effective Cache Space. While the effective cache space is shared among all concurrently running applications, the replication area is partitioned among the applications\(^2\). Whenever a data block is brought by an application from off-chip memory to the shared cache (effective cache space), a copy of it is also placed into the space allocated to the application in the replication area. The size of the replication area depends on the values of the target performance. That is, its size can change dynamically during the course of execution in order to satisfy the specified target performance value (which is, in our case, the overall hit rate in the effective cache area). Note that, this dynamic change is in general necessary because application data access patterns and cache space requirements can change from one phase of the execution to another. In other words, the same target performance value may require different amounts of cache spaces in two different phases during the course of execution. Therefore, our first control parameter is the size of the replication area which determines the size of the effective cache space, as illustrated in Figure 2.4. Simply put, if we reduce the size of the replication area, we can expect better performance but worse error resilience due to fewer replicas. In contrast, a larger replication area means better cache reliability (as we can have replicas for more data blocks) and relatively worse performance. Consequently, this control parameter has to be modulated carefully during the course of execution to satisfy the performance QoS.

Our second control parameter is the partitioning of the replication area among

\(^1\)In the ideal case RwR is 1, indicating that every block read has its replica ready in the replication areas. Note that, this does not necessarily mean that all the blocks currently in the caches have their replicas in the replication area (which is not possible anyway since the replication area is generally small, compared to the cache area where the original blocks are stored).

\(^2\)We use a “way partitioning” scheme similar to [62] to partition the shared cache among applications.
Effective Cache Space

Replication Area

1st Appl 2nd Appl 3rd Appl 4th Appl

Figure 2.4: Partitioning the shared L2 cache space into a shared area for storing original blocks and a replication area divided across applications. In our work there are two control parameters (knobs). The effective cache space is the main parameter to control the performance QoS. On the other hand, the per-application cache space allocation in the replication area is the control parameter to control the reliability QoS values.

Concurrently executing applications. Since each application normally exhibits a different data access pattern and data reuse than other applications and may have a different reliability QoS (specified target RwR value), the available replication area (whose size is determined based on the performance QoS) should be carefully partitioned across co-runner applications to satisfy their reliability QoS. It is to be noted that, in general, this partitioning is nonuniform (see Figure 2.4) and needs to be modulated at runtime to satisfy the specified RwR values. To summarize, we use the effective cache area size to satisfy the performance QoS of the entire workload and the replication area partitioning to satisfy the reliability QoS of individual applications. Both these control knobs must be modulated at runtime in order to satisfy the specified performance and reliability QoS on an epoch basis. In the rest of the chapter, we present a formal control theory based approach to this problem, and evaluate it experimentally.

Before moving to the discussion of the technical details of our scheme, we want to mention that there are other possible design alternatives regarding the management of the shared cache space. Specifically, as stated above, in this work, we allow the effective cache space to be shared among applications while partitioning the
Figure 2.5: Illustration of a simple feedback control loop for reliability management in a CMP.

replication area between them. Since effective cache space and replication area can be shared or partitioned, one has four potential design points. While most of the technical discussion below applies to all these four alternatives with appropriate modifications, we postpone a detailed treatment of the remaining three options to a future study. Note that, if the effective cache space is also partitioned, one can use other performance metrics for QoS specification like the weighted speedup metric [59] or harmonic mean of speedups [35], over a base scheme like equal partitioning to specify the QoS target.

2.4 Control Architecture

2.4.1 Background on Formal Feedback Control

Figure 2.5 illustrates a generic feedback control loop that can be used for the reliability management problem in a multicore. A control-theoretic approach to system management requires: (i) Identification of the criteria for optimization (e.g., minimizing the tracking error between actual performance and the reference target), which is the main function of the controller; (ii) Determination of the parameters to control (e.g., the cache allocation in the effective cache space, reliability area), called “control inputs”. The mechanism for the control is implemented in the actuator component shown in Figure 2.5; (iii) Monitoring of certain metrics to observe the effect of the control (e.g., the current hit rate or RwR metric), called the “out-
put variables”. This is the role of the *transducer* component (also seen in Figure 2.5); and (iv) A model to understand how much effect a change in the control parameters would have on the output variables (e.g., a differential (or difference) equation that characterizes the dynamic relation between the allocation of cache space and the hit rate), which is called the “dynamic system model”. Note that, the model is part of the controller.

### 2.4.2 Our Adaptive Control Design

Figure 2.6 shows a high level view of our control architecture. While our approach is applicable to any on-chip cache shared by applications, in our discussion, we
focus on a last level L2 cache (see Figure 1(b)). There are three main components of this control architecture: (i) A Performance Adaptive Controller (PAC) that is responsible for tracking the specified performance QoS in terms of the overall hit rate of the shared L2 cache; (ii) Per-application Reliability Adaptive Controllers (RACs) that are responsible for tracking the reliability QoS targets specified as RwR values per application; and (iii) the Actuator\(^3\) which is responsible for finding the best feasible partition of the last level cache space, considering the demands of all these controllers. Recall that, when we say a cache is partitioned, we mean that the different ways of the cache are assigned to different processors. We now describe each of these components in detail.

### 2.4.3 Performance Adaptive Controller

The Performance Adaptive Controller (PAC) controls the division between the effective cache space and the reliability area by determining the amount of the former required to guarantee the specified target performance QoS metric. PAC takes the target performance QoS metric as reference input. In this work, we considered overall hit rate of the shared L2 cache as performance QoS metric. As with any feedback control design, we need a model of the system. If the model is linear and not varying with time, simple classical controller designs like the Proportional Integral Derivative (PID) controllers are shown to be effective in achieving the control objectives [27, 32]. However, the cache behavior of applications has been shown to be highly non-linear, and dynamically varying [60]. In such cases, we would need a model with dynamically determined parameters and an adaptive controller that can use the dynamic model in order to achieve the control objectives.

The saturating non-linear behavior of variations on hit rate with the assigned shared L2 cache space can be modeled as an exponential function of the number of cache ways allocated. We model the shared L2 cache hit rate ($\lambda$) as a function of the number of cache ways allocated ($\omega$) as:

$$\lambda = \zeta^\infty(1 - e^{-\psi\omega}),$$

\(^3\)Note that, the Actuator component in our control architecture (Figure 5) performs more functions than just enforcing the inputs determined by the controllers as shown in Figure 2.5.
where $\zeta^\infty$ and $\psi$ are the model parameters that are determined at runtime. $\zeta^\infty$ is the hit rate in the shared L2 cache when it is allocated maximum number of cache ways (theoretically, $\omega = \infty$) and $\psi$ is a parameter that roughly determines the utility of each additional cache way that may be allocated to the shared L2 cache space. Note that, the model given in Eq. (2.1) very well captures the saturating behavior of the hit rate increase with the increase in the number of cache ways and saturates at a value of $\zeta^\infty$.

We use this dynamic model to design the PAC controller. The parameter being controlled in the PAC controller at time $t$ is $\omega(t)$ (the number of cache ways requested by the PAC controller to satisfy the performance QoS). We can write the general form of the control law for the PAC controller as:

$$\forall i, \omega(t) = \omega(t - 1) + \Delta\omega(t),$$  \hspace{1cm} (2.2)

where $\Delta\omega(t)$ is the correction computed for time $t$, by the controller for the current values of the model parameters. $\Delta\omega(t)$ can be determined using the difference between $\eta(t)$, the predicted value of $\omega(t)$ by the cache performance model, and the moving average $\mu(t - 1)$, of the previous values of $\omega$ determined using the same control law. Note that, the moving average is defined recursively over the values of $\omega$ computed by the PAC controller over the previous intervals of time as:

$$\mu(t - 1) = \alpha \mu(t - 2) + (1 - \alpha) \mu(t - 1).$$  \hspace{1cm} (2.3)

The older values of average ways ($\mu$) are exponentially attenuated by a factor $\alpha$, where $0 < \alpha < 1$. A higher value of $\alpha$ will increase the window size over which $\omega$ is averaged.

From Eq. (2.1), we can determine the value of $\eta(t)$ as predicted by the cache performance model by using simple algebraic manipulations as:

$$\eta(t) = ceil \left( - \frac{\log \left( 1 - \left( \frac{\lambda}{\psi} \right) \right)}{\psi} \right),$$  \hspace{1cm} (2.4)

where $\lambda$ is the target hit rate given as input to the PAC controller. Having determined all the individual components of the control law for the PAC controller, we
can now describe the control law (by consolidating Eqs. (2.2), (2.3), and (2.4)) as:

\[ \forall i, \omega(t) = \omega(t - 1) + \eta(t) - \mu(t - 1). \] (2.5)

We want to point out that \( \lambda \), the specified target performance metric should always be lesser than the \( \zeta^\infty \) as it is not possible for the controller to achieve a target performance better than \( \zeta^\infty \). If the reference hit rate of \( \lambda \) cannot be achieved, the system is said to lose its property of being controllable. In case of the PAC controller, it is easy to detect such a situation and also correct if necessary by using the minimum of the two values. It is clear that, we could also have a flexible admission control policy that accepts all workloads but either rejects or degrades those which over-specify target performance. Similarly, it is also not necessary that the system specifies the performance target as an absolute hit rate metric. With our model, we can also easily deal with higher level specifications like achieving \( x\% \) of the maximum hit rate achieved with the default shared cache.

2.4.4 Reliability Adaptive Controllers

Each reliability adaptive controller (RAC) is given a target in terms of the Reads-with-Replica (RwR) metric to be achieved for a given application. The control input (knob) in case of a RAC is the space allocated to an application in the reliability area. Note that, the total cache space available for the reliability area is limited by the minimum cache allocation for the shared L2 cache space as determined by the PAC to satisfy its performance QoS. The design of the RACs is based on the same principles as the PAC. However, the cache performance model is now maintained on a per-application basis. That is, each RAC maintains a cache performance model as shown in Eq. (2.1) that determines the relationship between the reliability area of the cache corresponding to that application and the RwR metric achieved for the reliability area. Each of the RACs then determines the amount of cache space required in its reliability area in order to satisfy the specified target RwR metric. The control law used by the RAC is derived in a manner that is very similar to Equation (2.5).

The concept of per-application RAC facilitates service differentiation in the degree of cache reliability requested by different applications, that is, each appli-
cation can specify a different RwR metric and this will be adhered to by the RAC controllers. Applications can make best use of this if they are aware of the application characteristics like the criticality of different data elements in how they reflect on the output and the access characteristics and degree of reuse of different data elements. Applications can estimate their reliability requirement in terms of the RwR metric by knowing these characteristics, and it is clear that each application will have a different requirement. Note that, we do not necessarily require the user or the application to be aware of the reliability requirements in terms of the RwR metric. If the application or the user specifies relative weights associated with the required level of reliability for each application, the system can abstract the presence of RwR metric. The actual RwR target values to be specified can be computed from the cache performance models maintained by the RAC in proportion of the weights and the maximum achievable RwR for the application as determined by the model. RAC can help applications realize these requirements without penalizing the overall performance QoS goal of the system.

2.4.5 Actuator

As explained earlier, the final cache demand placed by the PAC (ω) and the RACs are computed independently by each of them. It is feasible that the sum of the demanded cache spaces for the shared L2 area and the reliability areas is greater than the total available cache space. In such a case, we need an arbiter that determines the best possible allocation for the different areas such that all the constraints are met in the long term. In our architecture, all this logic is embedded in the Actuator component that decides the best allocation before enforcing it in the underlying cache.

Given a reasonable performance QoS specification, the cache space demanded by PAC for the effective cache space to satisfy the performance QoS is likely to be less than the total available cache space. As discussed earlier, we can easily detect the scenarios when the performance QoS is ill specified since λ, the specified target performance metric, should always be lesser always be lesser than the $\zeta^\infty$. In the pathological case that the performance target has been over-specified, in our current implementation, all the L2 cache space gets assigned to the effective
cache area, effectively eliminating the reliability area.\textsuperscript{4} We present a sensitivity study to the specified performance QoS targets in our experimental results.

The role of the actuator is more prominent when the performance QoS is satiable with fewer ways than that available in the overall L2 cache are assigned to the effective cache area. In this case, the actuator tries to satiate the reliability QoS values. If the sum of the demanded cache ways by the $N$ RACs ($\sum_{i=0}^{N} \omega_i$) is lesser than the cache space assigned for the reliability area ($\rho$), then each RAC is assigned its demanded fraction. The remaining cache ways ($\rho - \sum_{i=0}^{N} \omega_i$) may be annexed to either the effective cache space to improve the overall performance, or the replication area to improve reliability, or both. However, when the sum of the demanded cache ways by the RACs is greater than the available space for the reliability area, the additional $\sum_{i=0}^{N} \omega_i - \rho$ ways have to be spilled. One can have different strategies to handle this scenario. In our current implementation, these $\delta$ ways are recovered from each of the RACs in proportion to their demands in assigning the new values to the individual reliability areas ($\rho_i$) according to the following rule:

\begin{equation}
\rho_i^* = \text{floor} \left( \omega_i \left(1 - \frac{\delta}{\sum_{i=0}^{N} \omega_i}\right) \right). \quad (2.6)
\end{equation}

We determine $\rho_i^*$ as the floor, as we cannot exceed the total limit of $\rho$ in order to obtain a feasible partition. Other possible strategies include using application priorities to prioritize one application over the other in recovering these ways or a technique that would minimize the number of penalized applications by recovering all the ways from the most demanding applications. If there are any excess ways that remained due to this conservative estimation, they can be returned to the effective cache space to improve the performance of the system. Alternatively, one can distribute the remaining cache ways among the effective cache area and the reliability area (or both). In our implementation, we distribute the remaining cache ways among both the areas in proportion to their QoS specification.

\textsuperscript{4}An alternate approach would be providing at least a minimum cache space for replication area in any case. This alternate approach can be easily accommodated in our infrastructure.
2.4.6 Flow of Events

The measured hit rate of the effective cache space and the RwR metrics of the reliability area partitions are fed back into the controllers in order to complete the feedback loop. The PAC executes at a much coarser granularity than the RACs. That is, for each execution of the PAC, an effective cache area allocation is determined. This effective cache space allocation is kept constant for the successive executions of the RACs. After the completion of one interval of PAC’s execution, the parameters for the performance model of the effective cache area are recomputed with the newly-measured values of the observed hit rates. We use an interval of 100 million cycles for the execution of PAC and 10 million cycles for the RAC, that is, in each PAC interval, we have multiple (ten) RAC intervals. We later present sensitivity of our results to these parameters in Section 2.5.2. The actuator executes at the same granularity as the RACs as it needs to determine a feasible partition of the reliability area each time a new demand is placed by the RACs.

2.5 Experiments

<table>
<thead>
<tr>
<th>Processors</th>
<th>4 processors with private L1 data and instruction caches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor Model</td>
<td>in-order issue</td>
</tr>
<tr>
<td>Private L1 D–Caches</td>
<td>2-way set associative, 16KB, 64 bytes block size, 3 cycle access latency</td>
</tr>
<tr>
<td>Private L1 I–Caches</td>
<td>2-way set associative, 16KB, 64 bytes block size, 3 cycle access latency</td>
</tr>
<tr>
<td>Shared L2 Cache</td>
<td>128-way set associative, 4MB, 64 bytes block size, 10 cycle access latency</td>
</tr>
<tr>
<td>Memory</td>
<td>4GB, 260 cycle off-chip access latency</td>
</tr>
<tr>
<td>PAC Execution Interval</td>
<td>100 Million cycles</td>
</tr>
<tr>
<td>RAC Execution Interval</td>
<td>10 Million cycles</td>
</tr>
</tbody>
</table>

Table 2.1: Baseline configuration.
<table>
<thead>
<tr>
<th>Mix</th>
<th>Applications</th>
<th>Avg L1 miss rate</th>
<th>Avg L2 miss rate</th>
<th>IPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mix 1</td>
<td>h264ref, hmmer, calculix, dealII</td>
<td>0.079</td>
<td>0.013</td>
<td>1.45</td>
</tr>
<tr>
<td>Mix 2</td>
<td>astar, gcc, xalan, cactusADM</td>
<td>0.072</td>
<td>0.046</td>
<td>1.83</td>
</tr>
<tr>
<td>Mix 3</td>
<td>mcf, lbm, GemsFDTD, sphinx</td>
<td>0.202</td>
<td>0.257</td>
<td>0.74</td>
</tr>
<tr>
<td>Mix 4</td>
<td>soplex, omnetpp, mile, perlbench</td>
<td>0.105</td>
<td>0.062</td>
<td>1.82</td>
</tr>
<tr>
<td>Mix 5</td>
<td>bwaves, gromacs, sphinx, mile</td>
<td>0.118</td>
<td>0.038</td>
<td>1.56</td>
</tr>
<tr>
<td>Mix 6</td>
<td>perlbench, sjeng, omnetpp, mcf</td>
<td>0.137</td>
<td>0.027</td>
<td>1.87</td>
</tr>
<tr>
<td>Mix 7</td>
<td>GemsFDTD, sphinx, namd, povray</td>
<td>0.192</td>
<td>0.046</td>
<td>1.45</td>
</tr>
<tr>
<td>Mix 8</td>
<td>bwaves, gromacs, sjeng, perlbench</td>
<td>0.088</td>
<td>0.019</td>
<td>1.77</td>
</tr>
</tbody>
</table>

Table 2.2: Various mixes of the SPEC CPU2006 applications considered for our multiprogrammed workload. The third, fourth, and fifth columns show, respectively, the average L1 miss rate, L2 miss rate, and sum instructions per cycle of the different workloads when using the baseline configuration described in Table 5.1 without our scheme.

### 2.5.1 Benchmarks and Setup

The default multicore configuration used in most of our experiments is given in Table 5.1. We used SIMICS [44] and GEMS [45] to simulate this configuration and collect our experimental data. SIMICS is a tool-set that allows simulation of multiprocessor systems, and GEMS helps us to perform detailed (timing-accurate) core and memory hierarchy simulations. Since we perform cycle-accurate, full-system simulation, our results capture all the overheads incurred by the proposed scheme.

In this work, we used applications from the SPEC2006 benchmark suite [33] to evaluate the effectiveness of our scheme. We formed eight different workloads using these applications, as shown in Table 2.2. The last column of Table 2.2 gives (average) L1 and L2 cache miss rates for the workloads. Unless stated otherwise, each workload is simulated for 2 billion instructions (per core), after fast-forwarding the first 3 billion instructions. Throughout our discussion, we assume that there is one-to-one mapping between applications and cores. We use the `psrset` utility in Solaris to pin the applications to cores. In the discussion below, when we mention “hit rate”, we mean the hit rate in the effective cache space.
2.5.2 Results

Our first set of results present the RwR values under different sizes of the replication area when it is partitioned "equally" among four applications in a workload. Note that, this experiment does not use our proposed control-centric approach; instead, it just helps us study the trends when applications have equal replication areas.

Figure 2.7 plots the hit rates of different applications observed in the effective cache area when the effective cache area size is varied from a maximum of 124 cache ways to a minimum of 64 cache ways. We can make the following observations from these results. Firstly, some of the applications are more sensitive to changes in cache size than the others. For example, the hit rates of bwaves and gromacs in mix 5 varies by 19.6% and 14.5%, respectively, as cache allocations are varied from 124 ways to 64 ways. Secondly, the same application achieves different hit rates in different workload mixes even when the effective cache space is the same. For
Figure 2.8: RwR values with under different sizes of the replication area when it is partitioned equally among four applications in a workload. The configurations on the x-axis are the allocations for [effective: replica1: replica2: replica3: replica4], where effective is the cache ways allocated to the effective cache area and replica1 through replica 4 are the cache ways allocated to the four applications in the replication area.

example, when the effective cache space is 64 ways, in case of mix3 and mix6, mcf achieves hit rates of 36.0% and 41.9%, respectively. Figure 2.8, on the other hand, plots the RwR values of different applications when the replication area is equally partitioned among applications. We can make two observations from these results. First, as we increase the size of the replication area, RwR improves as can be expected; however in some cases the improvement is not significant beyond a point. For example, in Mix 1, increasing the replication area above 8 ways (2 ways per application) does not help with the RwR metric of applications. This is due to locality in data accesses, that is, in a given short period of time frame, only a couple of distinct data elements are accessed, and their replicas are maintained in the replication area. Our second observation is that different applications achieve very different RwR values with the same replication area size. For instance, with a replication area size of 8 ways, applications lbm and sphinx in workload Mix 3...
Figure 2.9: Performance targets achieved by our scheme. The numbers in the x-axis indicate the target performance QoS values.

Figure 2.10: Replication targets achieved by our scheme. A uniform target RwR value of 0.6 is set for all applications in all workloads.

Figure 2.11: Modulations in effective cache space to track the target performance QoS (Mix 7).

Figure 2.12: Tracking the hit rate by modulating the effective cache space (Mix 7).

Figure 2.13: Variations in instructions per cycle (IPC) by modulating the effective cache space (Mix 7).

Figure 2.14: Partitioning of the replication area (Mix 7).

achieve RwR values of 0.39 and 0.85, respectively. Adaptive feedback control based approach is the perfect tool for such dynamically varying workload characteristics. Note that, the RwR metric is a measure of reliability. Assuming that the cache errors are uniformly distributed, the RwR metric is a measure of the improvement in reliability obtained using our approach.

In our next set of experiments, we set the target performance QoS (hit rate) value, and target RwR value for each application, and use our proposed control centric scheme to dynamically modulate total replication area as well as its parti-
Figure 2.15: Tracking of RwR target for the four applications of Mix 7 under different sizes of the replication area when it is partitioned by our feedback controlled replication scheme.

The results plotted in Figure 2.9 reveal that our scheme is able to satisfy all QoS values specified for performance across all workload mixes. Note that, in our implementation, we first assign cache ways to the effective cache space, and therefore, the controller is able to successfully track any achievable QoS target with ease. Further, Figure 2.10 plots the RwR values achieved by different applications across workload mixes, when the workloads are run with the same performance QoS values as those used in Figure 2.9 and a uniform RwR target of 0.6 is specified for all applications in all workloads. It can be seen that, in most cases, the RwR metric is satisfied. However, in case of application mixes 2, 3 and 8, all applications do not achieve the reliability target as the PAC demands (and therefore consumes) most of the aggregate cache space for the effective cache area. As a result, replication area in these application mixes is affected. We later show that, when a reasonably lower performance QoS is demanded, all applications satisfy their reliability QoS demands.

### 2.5.2.1 Dynamics of our scheme

To better illustrate how our scheme is able to satisfy the specified performance QoS values, we present in Figure 2.11 how the partitioning between the effective cache area and replication area changes during the course of execution of Mix 7. It is easy to see that, as execution progresses, our approach modulates the boundary between these two areas by changing the effective cache space allocation to satisfy the performance goal (target hit rate value). Figure 2.12 plots the resultant hit
2.5.3 Sensitivity Studies

2.5.3.1 Varying the QoS specifications

The sensitivity of the PAC controller to the performance QoS specification in Mix 7 is shown in Figure 2.16. In this experiment, the performance QoS specifications are changed from 0.6 to 0.95. We can see that the PAC is able to meet the target until 0.8, but cannot meet the target when it is over-specified, beyond values from

Figure 2.16: Sensitivity to performance QoS targets.

rates with time for the same mix. We can see that the target hit rate of 0.7 is tracked with acceptable accuracy. Figure 2.13 also plots the performance of applications in terms of the overall IPC (sum of application IPCs). We can see that the IPC also gets stable as the hit rate gets tracked.

Figure 2.14 plots distribution of the replication area among four applications in the same workload. Again, as can be observed, our scheme modules this distribution dynamically in order to satisfy the specified reliability QoS at each epoch.

We now discuss the dynamics in the replication area. Figure 2.15 plots the tracking of RwR metric for four applications of Mix 7 when the performance QoS is set to a hit rate of 0.7 (i.e., 70%). We see that the specified RwR target of 0.7 is tracked with reasonable accuracy. Overall, these results clearly show that our approach successfully modulates the size of the effective cache area and partitioning of the replication area to satisfy specified performance and QoS targets.
0.85. Similarly, for a specific performance QoS value of 0.8 that the PAC is able to successfully track, we varied the RAC targets from a specification of 0.2 for all the applications to 0.9 for all the applications (see Figure 2.17). We observe a similar behavior in the sense that the RACs are able to track the specified QoS targets until a QoS specification of 0.6 for all applications is reached, beyond which it is unable to satisfy the over-specified targets.

2.5.3.2 Varying the controller execution intervals

The controller execution intervals for PAC and RACs are important design parameters. We studied the sensitivity of the controllers to track the reference inputs when invoked at different intervals. The results are shown in Figure 2.18. We observe that both the performance and reliability QoS targets are tracked for a range of execution intervals of the controllers, and hence the performance of the controllers is not highly sensitive to these parameters.

Figure 2.19: Performance of RACs when our scheme is used with a 16 KB L1 cache shared by four cores.

2.5.4 Results of our scheme when targeting L1 caches

So far, we have worked with a replication area for the L2 cache. However, our scheme is equally applicable at other levels of the cache as well, like the L1 cache. We studied the performance of the RAC controllers and their ability to track specified QoS targets for a 128KB L1 cache that is shared by 4 cores. The results for Mix1 to Mix5 with specified targets of 0.75 RwR for each application is shown in Figure 2.19. Additionally we observed that the RwR hit rates are generally higher in small L1 caches as the locality among L1 data blocks is generally higher.
2.5.5 Power Consumption

Power consumption overheads is an important concern when applying error protection techniques. In our proposed scheme, no extra delay is incurred compared to ECC. Reading of original and replica data can be performed in parallel. We have also compared power consumptions between our scheme and SECDED techniques. There is extra dynamic power overhead in our scheme due to read and write of replicas. However, as technology advances to 45nm and below, total power consumption is dominated by leakage power and dynamic power’s contribution is almost negligible. As indicated by our experiments, our approach and ECC consume about same amount of total cache power (average difference below 2%). In addition, caches equipped with ECC protection usually have a switch that can be used to turn on or off ECC. When applying our schemes, users can turn off ECC so no extra power is consumed.

2.6 Related Work

As process technology continues to scale down, managing reliability becomes an increasingly difficult challenge in the design of cache memories [28, 25] for high-performance microprocessors. Transient faults, also known as soft errors [48], occurring due to neutron and alpha particle strikes in large SRAM caches corrupt program data at the circuit level and may cause incorrect program execution and/or system crashes.

Using error correcting codes (ECC) [67, 66], which typically provide single-bit error correction and double-bit error detection and other similar schemes [39, 40] have been proposed previously. PADded Cache [53] is an orthogonal fault tolerance technique that uses a special Programmable Address Decoder (PAD) to disable faulty blocks and to re-map their references to healthy blocks. Duplication in memory for high availability was studied in [26]. As discussed earlier, all these schemes incur, higher performance, power, and area overheads. These overheads for L1 caches are quantitatively studied by Sadler and Sorin in [52].

The impact of propagation of hard errors [43] and soft errors [49] in the software has been studied in the past. Mukherjee et al [49] characterize these errors and
define a structure’s architectural vulnerability factor (AVF) as the probability that a fault in that particular structure will result in an error. A structure’s error rate is defined as the product of its raw error rate, as determined by process and circuit technology, and the AVF. They also identify numerous cases, such as prefetches, dynamically dead code, and wrong-path instructions, in which a fault will not affect correct execution. Several other previous works have modeled the impact of soft errors on combinational logic and on caches [64, 58, 61]. Kadayif et al [37] define a metric called AVFC (Architectural Vulnerability Factor for Caches) that represents the probability with which a fault in the cache can be visible in the final output of the program. Based on this model, they propose different architectural schemes for improving reliability in the existence of soft errors.

Replication [70, 68, 69] and selective data protection [42] for providing reliability in caches have been studied in the past. Zhang et al [70] proposed In-Cache Replication (ICR) for uniprocessors wherein the cache replicates data that is in active use within the cache itself while evicting those that may not be needed in the near future. This was extended in [68, 69] by proposing to add a small fully-associative cache to store the replica(s) of every write to the L1 data cache. The replicas can then be used to detect and correct soft errors. The replication cache can also be used to increase performance by reducing the L1 data cache miss rate. Providing reliability QoS has not received much attention so far. [63] propose a self-adaptive mechanism for cache reliability where a monitoring component monitors the error incidents within preset windows and sends the error information to a control component. The control component in turn, decides whether to replace the current reliability scheme or not among a set of reliability schemes. Feedback control theory has been used by some recent works in power [50, 65, 36] and performance management [60]. However, our work differs from these works as they do not consider reliability in the context of shared multicore caches.

2.7 Concluding Remarks

Reliability is an important concern with shrinking technologies. We propose a control theory centric approach designed to improve transient error resilience in shared caches while satisfying performance goals. The proposed scheme takes both perfor-
mance QoS and reliability QoS as inputs. We presented an experimental evaluation of the proposed scheme on various workloads formed using the applications from the SPEC2006 benchmark suite. The proposed scheme is able to satisfy, in most of the tested cases, both performance and reliability QoS by successfully modulating the total size of the replication area and partitioning of this area among the applications. The collected results also show that our scheme achieves similar results under different values of the major simulation parameters.
Performance and power consumption are important challenges faced by Network-on-Chip (NoC) designers. The situation is exacerbated when error control techniques are employed to provide reliability, since such techniques can lead to extra power consumption and execution cycles. In many systems today, ECC codes are used for error detection. Once an error is detected, recovery schemes are invoked to correct it. In this chapter, we focus on tuning error recovery schemes to explore performance, power and reliability tradeoffs. Previous reliability work targeting NoCs proposed two retransmission techniques to recover from errors: End-to-End retransmission and Hop-by-Hop retransmission. End-to-End retransmission can save power but can also incur longer delays for recovery by checking errors only at the destination. In comparison, Hop-by-Hop retransmission checks for errors at every router and has better performance at the expense of increased power overhead. We propose a novel retransmission scheme that employs feedback control theory to dynamically choose the time for error checking based on the performance requirements of the applications. Our scheme ensures that applications meet performance QoS and save power at the same time. Our experimental evaluation shows that, if a 10% slack in delay is allowed, our scheme can save as much as 80% of the power consumed by the underlying error control scheme. This chapter improves the design characteristic of many-core multiprocessor as shown in shaded part in Figure3.1.
3.1 Introduction

As the number of cores integrated into chips keeps increasing, global on-chip interconnection becomes a critical bottleneck in delivering performance under strict power constraints on many-core multiprocessors [76, 77, 78]. In such systems, traditional buses are not able to meet the challenge due to limited scalability and unbalance between gate delays and wire delays across the chip. One promising solution to solve this global interconnection problem is packet-based on-chip interconnection networks (NoCs) [76, 78, 79, 115]. An NoC based multi-core system consists of multiple point-to-point wire segments connected by intelligent routing blocks (routers). Once applications are mapped to the cores of a many-core multiprocessor, data packets can be relayed from any source node to any destination node, traversing multiple links and routing decisions are made in each router. With the advantage of high scalability, NoCs are considered the prime candidate to form the network infrastructure of future many-core multiprocessors. However, there are several challenges posed by NoCs, namely, performance, power and reliability.

One challenge faced by NoC designers is that of providing quality-of-service
(QoS) in performance. Packet-relay based NoCs help with the scalability problem but may incur in some cases longer delays compared to traditional buses. Specifically, data first needs to be split into structured packets/flits, appended with control information, and then, based on routing algorithms, transferred from one router to another until it reaches destination node. In a typical NoC router (Figure 3.2), data packets need to go through several stages: routing decision making, virtual channel allocation, switch allocation and link traversal. Limited capacity in router buffers also contributes to packet delays. These lead to unpredictable transmission latencies and difficulties to guarantee performance QoS. Various strategies have been proposed to provide guaranteed QoS on NoC based systems [97, 98].

Figure 3.2: A generic NoC router architecture with M PCs (Physical Channel) and N VCs (Virtual Channel).

Figure 3.3: Effect of load rate and error rate on performance.
NoCs provide higher communication bandwidth compared to traditional buses. However, they can also consume large amounts of power. In fact, power consumption has recently become one of the first-order constraints in NoC design [100]. It has been observed that, on existing multiprocessor prototypes, power consumed by NoCs can contribute up to 40% of the whole system power [116, 83]. Considering the fact that design of multiprocessors is already constrained by power/energy budgets, it is imperative for NoCs to go for power efficient options. Several strategies have been proposed to reduce power consumption such as DVFS and global clock gating. Such techniques are able to save power; however, one of the side effects is further reduced reliability (e.g., errors due to decreased voltage when voltage scaling is employed).

In addition to performance and power management, reliability is becoming a critical issue as we enter the deep sub-micron era. Errors occurring in NoCs can be categorized into permanent errors (hard errors) and transient errors (e.g., soft errors). Permanent errors are mostly caused by accelerated aging effects such as electromigration and manufacturing limitations. Transient errors, on the other hand, are mainly caused by aggressive shrinking of feature sizes of technology nodes. As devices shrink to the nanometer scale, the decrease in power supply voltage and device Vt leads to unreliable wires that are increasingly susceptible to all kinds of noises. Transient errors in an on-chip interconnect can be caused by many factors including crosstalk, electromigration, electro-magnetic interference,
alpha particles and cosmic radiations. It is critical for NoC designers to provide a certain level of resilience against such errors. There have been extensive studies in this area. For example, error detection and correction codes have been employed to protect against transient errors (e.g., parity, SECDED and FEC). Retransmission based schemes have also been proposed for error recovery. However, reliability is typically achieved at the expense of sacrificed performance and power. In particular, ECC CODECs and buffers used for error protection contribute to the power overhead. The power used for error correction can reach up to 20% of the overall network power in the current technology [104]. Performance is also compromised because extra cycles are needed for error checking and correction. Therefore, it is critical to fully explore the design space to build a reliability scheme with low performance and power overhead.

The goal behind this work is to employ feedback control theory to provide reliable data transmission, while minimizing the performance and power overheads at the same time. A feedback control-based approach is especially fit for this task due to two main benefits it brings: reference tracking and disturbance rejection. The controller can dynamically track the system output and ensure that the output follows the changes in reference input by adjusting system parameters. A feedback controller can also minimize external disturbances which helps to satisfy the target QoS. The main contributions of this work include:

- We propose a dynamic retransmission based scheme that incorporates the advantages of both End-to-End and Hop-by-Hop schemes. In our scheme, the number of hops to perform ECC checks is not fixed but adjustable based on the current network state and the specified (application-level) QoS.

- We demonstrate how a feedback controller can be employed to dynamically decide the number of hops at which to check ECCs. We also demonstrate how to model the entire system to build such controllers.

- We present a detailed experimental evaluation of our proposed scheme. Our results reveal that, by dynamically adjusting ECC checking intervals through feedback controllers, a significant amount of error control power can be saved whenever performance slack can be exploited.
The rest of the chapter is organized as follows. In Section 3.2, we describe the state-of-art NoC reliability schemes. In Section 3.3, we explore the design space for exploring the reliability, power and performance tradeoffs. We also present our dynamic retransmission based reliability scheme. Section 3.4 discusses the system modeling and controller design. Our experiment evaluation is presented in Section 3.5. Section 3.6 briefly goes over the related work. Finally, we conclude the chapter in Section 4.6.

3.2 Error Control on NoCs

Errors that occur in an NoC-based system at run time can be categorized into hard errors and transient errors. In this work, we focus on transient errors, which can be further categorized into link errors and Single Event Upsets (SEUs). Link errors include crosstalks and coupling noise which are mainly caused by aggressive technology scaling. SEUs on the other hand are caused by cosmic rays and alpha particles [103, 107, 94]. Among the transient errors, link errors are the most prominent problem [103]. SEUs are rare and usually affect only one bit of data; consequently, NoCs equipped with SECDED ability are good enough to handle them. Unlike the SEUs, however, link errors such as cross talks often involve multiple bits of data. In such cases, SECDED does not have the power to correct the data. More powerful error correction codes such as Double Error Correction Triple Error Detection (DECTED) are typically too expensive to implement with power and area concerns. As a result, when multi-bit errors are detected, retransmission is usually employed [104, 108, 109].

As stated earlier, there are two types of error control schemes: End-to-End (E2E) and Hop-by-Hop (HbH). In the E2E scheme, data is encoded by ECC encoders at the source node before being sent out to the network. No ECC checking is performed until the data reaches its destination node. If an error is found, the destination node sends a nack to the source node, and the data gets retransmitted. In this scheme, a correct copy of the data is stored in the source node retransmission buffers until the data reaches the destination without any error. The E2E error checking can be performed based on units of packet or flit. In comparison, in the HbH scheme, data is encoded at the source node first, then on its way to
destination, at every node it passes, ECC decodings and encodings are performed on the data. If there is no error, then data hops to the next node on the path, and is also saved to the retransmission buffer in current node. The backup data is removed from the retransmission buffer when the next node receives the data correctly. Unlike the E2E scheme, the two copies of data are always kept in two adjacent nodes in the network. In high performance NoC router designs, ECC encoding/decoding operations are performed in parallel with other pipeline stages inside a router, so there is no extra delay caused by ECCs unless an error is found. In this work, we assume that the NoCs under study employ this kind of ECC design.

These two error recovery schemes have been thoroughly analyzed and compared in previous studies [104, 106, 94, 95]. Each of them has its advantages and drawbacks. Specifically, the HbH scheme can find the error at the earliest point and start retransmission right away. Consequently, the retransmission path is short and the impact on packet delay is minimum. The main drawback of this scheme is its high energy overhead. This is because, at every hop, ECC decoding and encoding are performed. In addition, data needs to be sent to the retransmission buffer which can consume significant amount of power [108, 109]. In the E2E scheme, energy consumption is comparatively lower because ECC operations are only performed at the end points. However, the impact on performance may be high since the data that is resent needs to traverse the whole path from source to destination. As a result, the average packet delay is typically higher and this approach puts more pressure on network traffic.

Figure 3.3 and Figure 3.4 illustrate how performance and power are affected by these schemes. In Figure 3.3, each curve indicates the average packet delay with fixed error rate under one retransmission scheme (either E2E or HbH). We vary the load rate (flit/cycle/node) from 0.1 to 0.4. As is shown in the plot, in both schemes, the observed delay increases when the load rate increases. Higher error
rates also incur higher data delay due to increased traffic for retransmission. When
the error rate is very low, E2E and HbH have similar performance. However, E2E
performs significantly worse when error rate increases to a certain level. Figure 3.4
shows that HbH scheme consumes more energy than E2E scheme. The difference
in power between the two schemes gets lower when the error rate increases. This
is due to the higher power consumption for every retransmission in E2E scheme
compared to HbH scheme.

3.3 Power and Performance Tradeoffs

Based on the analysis of these two data retransmission schemes, we propose a novel
scheme that incorporates the advantages of both, while avoiding their drawbacks
at the same time (Figure 3.5 illustrates the tradeoffs that can be explored). The
method we propose is to dynamically decide number of hops to invoke ECC during
execution. Its implementation is similar to that of the E2E scheme; however,
the distance between the correct copy of data and transmission destination is
adjustable. For example, if we decide that the hop count at which we check ECC
is three, we perform ECC checks in every third hop along the path. No ECC
is performed at routers within three hops after data is already checked. As a
result, we can detect errors earlier and shorten the retransmission path to meet
QoS requirements. We can also reduce the energy consumed by ECC checks as
we reduce the number of encoding/decoding operations and retransmission buffer
writes.
Consider Figure 3.6, which shows a scenario where one application is sending data from core 5 to core 23. If the E2E scheme is applied, data is only checked at core 23 for correctness. If the HbH scheme is employed, at all nodes along the path (i.e., nodes 6, 7, 8, 13, 18, 23), an ECC check will be performed. In our proposed scheme, the ideal number of hops in which to perform ECC checks is automatically determined based on the user-specified QoS. For example, when the performance requirement is not tight, this number of hops can be relaxed to 6, same as in the E2E case to save the power consumed for error control. However, when the application requires shorter delays, ECC hops can be reduced to meet performance requirement, e.g., every 3 hops. In this case, the ECC will be performed at nodes 8 and 23 only for our example scenario. As a result, the error control energy increases, but the retransmission path is shortened, leading to shorter delays.

![Figure 3.7: Hardware modification on ECC decoding.](image)

![Figure 3.8: Hardware modification on ECC encoding.](image)

The modification needed to implement this scheme is small. We only need to add a field in the header flit of a data packet as hop counter after last ECC check (as shown in Figure 3.9). For example, the counter is initialized to 3 before being sent to the network. The counter is decremented at each router (hop) on the path.
Every router checks this field first, when it is not zero, no ECC check is performed on all the flits of this packet, and data is directly forwarded to next stage in the pipeline. If a router finds this field to be zero, an ECC check is performed. If no error is found, this field will be reset to three and the flit can go to the next hop. Figure 3.7 and Figure 3.8 illustrate how the hardware can be modified to implement our proposed scheme. ECC CODECs and retransmission buffers can be supply-gated in order to further eliminate leakage power. Compared to the E2E and HbH schemes, the main advantage of our scheme is that the power consumed by the error control can be reduced as much as possible by fully exploiting the performance slack. Since the on-chip network traffic is dynamically changing and each application typically has its own performance requirement (QoS), our proposed scheme is more flexible to adjust to such an environment. In the next section, we describe in detail how we use the feedback control theory to implement our scheme.

3.4 Feedback Controller Design

3.4.1 Feedback Control Theory

Feedback control provides a way to use measurements of a system’s outputs such as response times, throughput, and utilization to achieve externally specified goals by adjusting the system control inputs/parameters (e.g., buffer sizes, scheduling policies, priorities). The appeal of feedback control is that we can achieve the desired output (QoS) by directly specifying the reference input instead of manipulating the control input indirectly [106]. Another advantage of feedback control is its ability to ensure that the measured output tracks the reference input even in the presence of disturbances. Almost any automatic system has some feedback control elements. The concepts of feedback control apply to computing systems as well.
In fact, there have been many recent efforts in applying feedback control theory to various aspects of computer systems, including load balancing [106], streaming media [106], power management [110] and QoS management [97]. In this work, we apply feedback control theory to achieve performance-power tradeoffs on NoCs that employ ECC to provide reliable intra-chip communication. To apply feedback control approach, the following steps need to be executed: (i) identification of the optimization criteria; (ii) selection of the parameters to control; (iii) observation of the effect of the control output; and (iv) construction of a system model to characterize the target system behavior.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
<th>Corresponding parameter in our design</th>
</tr>
</thead>
<tbody>
<tr>
<td>reference input</td>
<td>the desired value of the measured outputs</td>
<td>the desired packet delay given by the QoS</td>
</tr>
<tr>
<td>measured output</td>
<td>a measurable characteristic of the target system</td>
<td>the measured packet delay of the NoC under study</td>
</tr>
<tr>
<td>control error</td>
<td>a parameter that affects the behavior of the target system and can be adjusted dynamically</td>
<td>the number of hops for ECC checks</td>
</tr>
<tr>
<td>control input</td>
<td>a parameter that affects the behavior of the target system and can be adjusted dynamically</td>
<td>the number of hops for ECC checks</td>
</tr>
<tr>
<td>controller</td>
<td>a hardware or software implementation that determines the setting of the control input needed to achieve the reference input. The controller computes values of the control input based on current and past values of control error.</td>
<td>a PI controller implemented in software</td>
</tr>
<tr>
<td>target system</td>
<td>the computing system to be controlled</td>
<td>an NoC being investigated</td>
</tr>
</tbody>
</table>

Table 3.1: Control system components and corresponding parameters in our design.
3.4.2 Feedback Control System Components

Figure 3.10 shows a high-level view of a generic feedback control system. The essential elements of a feedback control system and the corresponding parameters in our specific design are described in Table 3.1.

3.4.3 System Modeling

The design of feedback control systems requires the ability to quantify the effect of control inputs on measured outputs. Developing an accurate system model is one of the most important parts. There are several methods to construct a system model, including static modeling, statistical modeling, and black-box modeling.

Choosing the right method can greatly increase the model accuracy and simplify the overall modeling process. In this work, we have chosen the black-box approach to construct our system model. The appeal of the black-box model is that it requires a less detailed knowledge of the relationships between inputs and outputs. The only things that need to be known are the inputs and outputs of the target system. Based on these, we can infer their relationships by applying statistical techniques to the data collected.

The NoC system under investigation can be expressed using an ARX model [106]. The general form of an ARX model is

\[
y(k + 1) = a_1 y(k) + \cdots + a_n y(k - n + 1) + b_1 u(k)
\]

\[+ \cdots + b_m u(k - m + 1).\]

In this equation, \(y(i)\) is the system output and \(u(i)\) is the system input. Because our next output depends only on the inputs and outputs from one time unit in the past, we can employ a simplified version of the ARX model, referred to as
the first-order model. The first-order model can be described by the following equation:

\[ y(k + 1) = ay(k) + bu(k). \]

The above equation provides a way to predict \( y(k+1) \) from \( y(k) \) and \( u(k) \). We use \( \hat{y}(k) \) to denote the predicted value. We have:

\[ \hat{y}(k + 1) = ay(k) + bu(k). \]

The prediction error is then:

\[ e(k + 1) = y(k + 1) - \hat{y}(k + 1). \]

We collect data by providing inputs and observing the outputs. In order not to bias the parameters of the black-box models, data should be collected uniformly across the input range. In our case, the input data is the number of hops at which ECC is to be checked. Therefore, we linearly increase the number of hops from 1 to the entire length of the communication path. We then apply least-square regression to get the values of \( a \) and \( b \) so that the sum of square errors is minimum. After setting up the model parameters, we need to evaluate the accuracy of the model. In our experiments, we ensure that the error is within the range of 0-4%.

### 3.4.4 Controller Design

We choose a PI controller as our feedback controller. A PI controller consists of two parts: proportional controller and integral controller. The proportional controller can reduce the steady-state error, but cannot drive the error to zero. However, this can be achieved with the help of the integral controller. By incorporating the two controllers together, we can effectively reduce the steady-state error to the minimum. Another reason why we chose a PI controller is because it is simple to implement, and consequently, the resulting overhead is low. A PI controller has the form of:

\[ u(k) = u(k - 1) + (K_P + K_I)e(k) - K_P e(k - 1), \]

where \( u(k) \) and \( u(k - 1) \) are the number of hops for ECC checks. \( e(k) \) and
\( e(k - 1) \) are the errors between the reference input and the measured output of the current and last time units. After deciding \( K_P \) and \( K_I \), we can derive the next hops count (following which an ECC check is to be performed) from the current hop count and the previous system errors. Different methods can be used to determine value of \( K_P \) and \( K_I \). In our work, we use Pole Placement which is described in detail in [106] to obtain those parameter values.

### 3.4.5 Overheads

We now discuss the overheads introduced by our feedback control-based system. Our control system is implemented as software applications running on one core or multiple cores. The hardware overhead is negligible: the only hardware support needed is to measure system output, i.e., the data packet delay. In an NoC based system that provides QoS services, such mechanisms should already be in place. As described above, the software implementation is also very simple, incurring little overhead to the system performance.

### 3.5 Experimental Evaluation

In this section, we present our experiment setup and discuss the results collected. We ran simulations using an in-house cycle accurate NoC simulator and injected 300,000 packets into a 5 × 5 NoC. Our NoC simulator applies state-of-the-art techniques such as low latency pipelines, virtual channel flow control, input buffering, ECC encodings/decodings, and retransmissions triggered by errors. Uniform traffic is used in all our simulations. We assume that our routers run at 2GHz. We use CRC as our ECC, and we derive energy consumption in the routers using a combination of CACTI 6.0 [148] and Orion [105] in 50nm technologies. Static power consumption is also considered. Table 5.1 and Table 3.3 list respectively the NoC setup and major power parameters used in our experiments.

In our experiments, we evaluate an NoC using the following two metrics: average latency per error-free packet for performance and average energy per error-free packet for power. We performed experiments with different network traffic and reliability scenarios. We demonstrate, in each case, how our feedback controller
Table 3.2: NoC Network Configuration.

<table>
<thead>
<tr>
<th>NoC Topology</th>
<th>2D Mesh</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoC Dimension</td>
<td>5x5</td>
</tr>
<tr>
<td>Switching Technique</td>
<td>wormhole</td>
</tr>
<tr>
<td>Packet Size</td>
<td>4 flit</td>
</tr>
<tr>
<td>Flit Size</td>
<td>256 bit</td>
</tr>
<tr>
<td>Routing Algorithm</td>
<td>X-Y routing</td>
</tr>
</tbody>
</table>

Table 3.3: NoC Component Power Consumptions(mW)

<table>
<thead>
<tr>
<th>Component</th>
<th>Power Consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Routing</td>
<td>1.7</td>
</tr>
<tr>
<td>Buffers</td>
<td>16.32</td>
</tr>
<tr>
<td>Crossbar</td>
<td>11.97</td>
</tr>
<tr>
<td>ECC Codec</td>
<td>0.52</td>
</tr>
<tr>
<td>Retransmission Buffer</td>
<td>3.83</td>
</tr>
</tbody>
</table>

can adapt to the current NoC traffic condition and achieve performance and power tradeoffs by dynamically tuning the number of hops per ECC check.

3.5.1 Case 1: Fixed Load Rate, Fixed Error Rate

In the first case, we fix error rate (0.0001) and packet load rate (0.35). When the desired QoS delay is 30 cycles, the feedback controller can relax the ECC checking interval to every 5 hops (Figure 3.12). As shown in Figure 3.11, the observed delay...
is indeed tracking the desired reference, and energy consumption for error control is reduced to the minimum (Fig. 3.13).

### 3.5.2 Case 2: Fixed Error Rate, Changing Load Rate

One of the dynamic factors affecting the performance of data transmission in an NoC is the changing load rates. In the second case, we fix the error rate and change the packet load rate in the middle of simulation from 0.1/flit/node to 0.4/flit/node. Figure 3.14 and Figure 3.15 show how the feedback controller tracks this change and update the hops for ECC checks. Initially, 5 hops is good enough to satisfy
the delay QoS. However, at time 22, when load rate increases, the delay increases to 33 cycles if ECC is still performed every 5 hops. Consequently, the feedback controller reduces the number of hops at which to check ECC to 3, in order to ensure the average delay is below the reference (31 cycles). Figure 3.16 shows how much energy can be saved by this change.

3.5.3 Case 3: Fixed Load Rate, Changing Error Rate

In reality, error rate can change dynamically too. For example, more errors may be found when a chip is overheated or when more external noises occur in its working environment. In this case, we fix the load rate at 0.35 and change the error rate in the middle of simulation from 0.001 to 0.005. Figure 3.17 and Figure 3.18 show how the feedback controller can respond to this change and update the hops for ECC checks correspondingly. Initially, at error rate of 0.001, ECC checks are performed once every 5 hops to satisfy the delay QoS. However, at time 22, the error rate rises to 0.005, if the ECC hops remains 5, the delay would increase to 38 cycles, which is above the desired QoS. So, the controller reduces the number of hops to 1. As in Figure 3.19, the energy consumption increases but we achieve the desired performance QoS.

3.5.4 Case 4: Multiple Applications with Overlapping Routes

In this case, we perform experiments with multiple applications whose transmission paths overlap with one another. Initially, only one application is running (node 5 to node 18), and the controller selects an optimal hops (5) to perform ECCs in an attempt to save energy and also meet the delay QoS. Later, a second application starts sending data and shares some path with the first application (node 6 to node 9). Accordingly, the feedback controller reduces the ECC hops in order to track the delay requirement. Some time later, a third overlapping application starts running (from node 8 to 13) and the controller further reduces ECC hops. Later when both application 2 and application 3 finish and the traffic load on the shared path is reduced, the controller increases the ECC hops in order to save energy. Figure 3.20, Figure 3.21 and Figure 3.22 show the delay, ECC hops and energy consumption changes that take place during execution.
3.5.5 Comparison of Power Overhead using Different ECC Coding Techniques

We now expand our design space exploration by investigating the power overheads brought by different ECC coding techniques. Among the various ECC coding techniques that can be applied to protect against soft errors, we choose the most commonly-used ones: CRC, SECDED, and DECTED. CRC codes are usually used for error detection, so they have the lowest error protection power. However, they also result in the lowest power overhead. SECDED can correct one-bit errors and
detect two-bit errors, but the power overhead is increased significantly compared to the CRC codes. SECDED can be implemented using Hamming codes or Hsiao codes. DECTED codes are more sophisticated ECC codes that can detect three-bit errors and correct two-bit errors, but with the highest power overhead among all three techniques. In our experiments, we explore the power overhead when the error rate changes between 0.00001 and 0.002 (the error rates here are uncorrectable error rates). With each error rate, we observe the power overhead when ECC is checked at every hop or only once at the destination, which gives the lower bound and upper bound of power consumption for the error control. Figure 3.23 gives the average packet energy consumptions. Figure 3.24 shows the power overhead of error control using different ECC coding techniques. The figures show that the CRC technique...
consumes the least amount of power compared to SECDED and DECTED (below 5%). DECTED can consume up to 20% of overall power consumption. On the other hand, SECDED and DECTED can achieve better performance compared to CRC if the errors detected can be recovered by their correction ability. For example, one bit error can be corrected by SECDED codes, thus no retransmission is incurred. However, for multiple bit errors (such as link errors), SECDED codes are not powerful enough to correct them, so data has to be recovered by retransmission. If the DECTED/SECDED schemes are employed for error control, data should better be checked for errors with larger number hops because more power can be saved. Since the ECC codes with low error protection ability (e.g., CRC or parity) incur relatively small power overheads even when ECC is checked at every hop, ECC checks should be probably performed more frequently, in order to improve performance in such schemes. Note that our feedback control scheme works regardless of which ECC technique is employed. Under a specific ECC technique, as long as we have a reference input specified, our proposed scheme can find an optimal number for ECC checks that can save power and provide QoS guarantee at the same time.

3.6 Related Work

There have been extensive studies on NoC error control schemes [108, 109, 93, 94, 95, 96, 98]. Kim et al [108] presented a flit based Hop-by-Hop retransmission scheme to handle uncorrectable errors. In this scheme, on every hop along the path, data needs to be checked by ECCs and written to retransmission buffers of three flits deep. The scheme can detect errors and correct them on the earliest occurrence; the down side is high power consumptions for error control on every hop. This scheme is best suited for environments with high reliability requirements.

Yu et al [93] proposed an adaptive error control method for NoCs running in a variable noise environment. Nodes in the NoC have multiple level error control abilities. ECC schemes of different correction abilities are selected based on error rates. When the error rate is low, a simpler scheme is used to save power. If the error rate increases, a more complicated ECC scheme is chosen to increase reliability. The drawback of this scheme is each node needs to be implemented
with multiple ECC functional units, increasing the hardware overhead.

Based on the observations that some applications have inherent error tolerance, Yanamandra et al [107] proposed a scheme to exploit the AVF factors in order to save power while ensuring certain reliability levels. Since not all transient faults result in errors in program outputs, error correction mechanisms are selectively turned off in their scheme to save power.

Feedback control theory has been applied to various aspects of computer systems. Sharifi et al [97] employed feedback control to provide QoS in NoC based systems. Based on reference throughput, the controller dynamically allocate virtual channels to each application to meet the QoS requirements. Ogras et al [110] proposed a feedback control based technique for NoCs consisting of multiple voltage-frequency islands to cope with power consumption, clock distribution and parameter variation problems.

In comparison to these prior studies, our goal in this work is to present and evaluate an adaptive strategy which modulates the number of hops at which to check ECC based on current network traffic and specified performance QoS.

### 3.7 Conclusion

In this work, we present a novel error control scheme for NoC based many-core multiprocessor systems. Two key observations lead to our work: (a) Hop-by-hop retransmission scheme can recover from errors quickly but with high power consumption. End-to-end retransmission can save power but leads to longer data packet delay which compromise QoS services. (b) On line traffic is dynamically changing so neither scheme is flexible enough to achieve the best performance and power tradeoffs. Feedback controller can adjust to such changes and keep track of reference inputs. Our feedback control based scheme dynamically decides when ECC checks are performed based on current network traffic. Our experiment results show that our proposed method can adapt to changes in load rate, error rate and multiple applications running simultaneously. As long as the specified performance QoS allows, our controller will choose the optimal setup to reduce error control power consumption to the minimum.
Chapter 4

Exploring Performance-Power Tradeoffs in NoC Design with Heterogeneity

4.1 Introduction

In recent years, chip multiprocessor based systems have become the focus of attention in computer architecture design. Designers are pushed by Moore’s Law to integrate a large number of cores in a single chip. In fact, companies such as Tilera and Intel have already developed prototype CMPs with as many as 100 cores integrated in one silicon chip [100, 101, 103, 104]. At this density, traditional buses are no longer able to serve as the backbone of on-chip interconnections and are projected by many to be replaced by Network-on-Chips (NoCs). NoCs evolve from the off-chip networks that connect multi-processor systems. As interconnection networks shift to on-chip environments, designers face new challenges. Routing latency has become a more prominent factor affecting overall performance. At the same time, power consumed by NoCs and the area occupied by NoCs have become more expensive overheads in an on-chip environment. As a consequence, a low cost NoC design that can provide competitive performance has become a focus.

Among an NoC router’s components, buffers consume a significant fraction of the power and area. Routing latency and design complexity also increase when buffers are employed in a router. Thus, removing buffers to reduce the cost becomes an appealing option. Several recent works have investigated ways to totally or
partially remove buffers from a router \cite{107, 108, 109}. In particular, Moscibroda and Mutlu \cite{107} proposed a bufferless NoC framework (BLESS) that removes input and output buffers from the router. If contentions occur in an output port of a bufferless router, data packets are deflected to neighboring nodes using hot-potato routing. However, several issues remain unaddressed by their bufferless NoC scheme. First, BLESS only works well when the network load ranges from low to medium. If the network utilization is high, network performance degrades sharply with the increase in router output contentions. Second, since data packets cannot be stored in a router’s buffer, they have to keep travelling in the network until they can finally reach the destination. Third, differentiated service classes are not supported since there are no buffers to keep the data from moving towards the destination.

In this work, we propose a novel heterogeneous NoC architecture that solves these problems by employing both buffered and bufferless routers in a same NoC. We first investigate how the use of these two types of routers in the same network can affect NoC performance and cost. We show that, based on user’s performance and power requirements, designers can select an optimal placement plan that can strike a balance between cost/power and performance. We next investigate how application mapping can provide differentiated service among the applications and propose a Buffered Router Aware Mapping (BRAM). Finally, we propose a Buffered Router Aware Routing (BRAR) algorithm that can route data packets along buffered routers in order to further improve performance. We show that our proposed techniques can significantly increase network performance under high network loads and meet the cost constraints at the same time.

This characteristic of many-core multiprocessor improved by this chapter is shown as shaded part in Figure 4.1.

### 4.2 NoC Design with Bufferless Routers

NoCs with bufferless routers are proposed to reduce the power and area overhead of on-chip interconnections. BLESS \cite{107} is a bufferless NoC framework that employs deflection routing. In BLESS, data can be routed in flits (flit-bless) or worms (worm-bless). We implemented the flit-bless scheme as our baseline architecture.
Because there are no buffers in the routers to store flits, data coming from an input port has to be routed out through one of the output ports to avoid being lost. When contention occurs due to multiple in-coming flits competing for a same output port, only one flit is allowed to use that output port based on priority policies. Other flits are intentionally mis-routed. In order to guarantee all data flits can finally reach their destinations, the routers must have at least the same number of output ports and input ports. In addition, since there is no buffer in the routers, a processor can inject a flit into its router only when at least one incoming port is free. Our baseline BLESS router consists of two stages: route calculation (RC) and switch traversal (ST). In our evaluations, we assume each stage takes one cycle, so it takes in 2 cycles in total for a flit to pass through a router.

Prior proposed bufferless router designs apply two major techniques to handle contention: deflection [107, 117] and retransmission [108, 109]. In a deflection (intentional mis-routing) based bufferless NoC scheme, data packets are routed to an available port, which may not be the optimal port selected by the underlying routing algorithm, when contention occurs. Packets to be deflected are selected
Based on priority to avoid deadlock and livelock. Consequently, since data cannot be saved in a buffer, if a packet is not able to use its optimal port, it bounces around in the network until contentions diminish or it becomes old enough to win the arbitration. In comparison, in a retransmission-based bufferless NoC scheme, data packets that fail in port contention are simply dropped and will need to be retransmitted. Thus, extra messages are needed to pass the information about whether a transmission is successful and, if not successful, to retransmit the data. Both schemes have long data latencies when the network load is high. In addition, large amounts of power can be consumed by deflected or retransmitted data. This motivated us to investigate the design of a network that has buffers in some of its routers to keep the packets from being deflected or dropped. In this way, data packets can avoid travelling extra links on their path toward their destinations. Power consumption can also be reduced when packets are held in buffers in some nodes in the network rather than being routed in a non-optimal direction.

### 4.3 Our Approach

Based on the fact that the BLESS NoC framework can greatly reduce the cost of a router, but cannot deliver very good performance at high network loads, we propose a heterogeneous NoC design that employs both buffered and bufferless routers in the same on-chip network. Our bufferless routers are similar to the routers in BLESS: input buffers are completely removed from each input port. However, in our buffered routers, we use buffers to store incoming data. There are two pipeline stages in the routers: RC and ST. We use the deflection algorithm to route data flits in both types of routers. When several flits are competing for the same output port, the oldest flit will be granted the port. In a bufferless router, an incoming flit will join the arbitration immediately after passing through the RC stage. In a buffered router on the other hand, the first flit in the buffer queue is selected to be routed and sent to arbitration if the buffer queue is not empty. A new flit can bypass buffer writing when the buffer is empty. Flits failing in the competition are treated differently by the two types of routers. In a bufferless router, the flit with lower priority will be deflected to an available output port. In a buffered router, if the buffer is not full, instead of being deflected, the data flit
stays in the buffer queue (the head of queue remains in the first position). If the buffer is full, then the head flit in the queue needs to be deflected like in a bufferless router. The basic principle of deflection routing still applies to buffered routers: arbitration is determined based on flit ranking. The benefit brought by buffered routers in our design is to reduce the probability of misrouting. In the next cycle, the head flit in the buffer queue gets a second chance to compete for its desired output port. The possibility that a data packet is routed to a productive output port is increased by storing them in the buffered router. Our design is deadlock and livelock free: a data flit will not be travelling forever in the network or stuck in a buffer. Since we adopt age as our policy for flit ranking, once a flit becomes old enough it will get hold of the requested ports and reach the destination.

There are two advantages brought by this mechanism. First, buffers can reduce data transmission latency by keeping a flit close to its optimal path. Figure 4.2 illustrates a flit that travels from node A to node F. When applying X-Y routing algorithm, the flit will be allocated to the East output port. If there is another flit with higher priority demanding the same output, this flit will be deflected to some other port. If the North output port is available, then this flit can still travel on an optimal path. We call this case an optimal deflection. However, if the network load is high, the chances that this flit can pass through port North are decreased and, as a result, the flit travels further away from its destination (either West or South). For each non-optimal deflection, the minimum delay incurred, i.e., the time for the flit to come back to the shortest path can be calculated as

\[
Latency_{\text{deflect}} = Latency_{\text{link}} \times 2 + Latency_{\text{router}}, \tag{4.1}
\]

where the \( Latency_{\text{link}} \) is the time in cycles used to traverse a link, and
Latency\_router represents the number of cycles spent in the router pipeline. In our baseline NoC framework, the Latency\_deflect is at least 4 cycles: one cycle to a neighboring node, two cycles to get out of that node, and one cycle to come back to a node on the optimal path. A buffered router can reduce this latency. For example, if the output port to the East or North becomes available in the next cycle, it takes a buffered router 1 cycle of delay to enter the RC stage again compared to 4 cycles when it is deflected. Second, buffers in some regions of the NoC can absorb deflected flits and stop the traffic contention from spreading to other regions of the network. The functionality of buffered routers is similar to that of a reservoir that stores water to protect the surrounding area from being flooded. In the pure bufferless NoCs, there is no mechanism to control such data flooding. Data communication in one region of the network can easily get affected by that of another region.

4.3.1 Router Placement

We assume a 2-D mesh on-chip network that has 8×8 nodes, each with a radix of 5. Our routers employ deflection routing. We place buffered routes symmetrically because our proposed NoC is for general purpose CMPs. Our goal is to investigate the effect of placement of heterogeneous routers on performance, power and buffer overheads. Figure 4.3 illustrates the placements studied in this work. The light colored nodes represent bufferless routers and the dark colored nodes represent buffered routers. For each placement, the ratio of buffered routers and bufferless routers is also shown as the number of buffered routers : the number of bufferless routers. Note all routers in P8 are buffered but employ deflection routing. Since our motivation is to reduce traffic contentions by placing more powerful routers in the network, it does not make much sense to put the buffered routers along the edges of the network.

In our first category of placements, buffered routers are placed to the center region of the NoC, based on the intuition that the center of the mesh tends to be the hot spot of traffic congestion. P1, P3 and P7 fall into this category of placements with 4, 16 and 36 buffered routers respectively. Our second category of placements use buffered routers so that the whole network can be divided into
several regions in order to mitigate the effect of deflected traffic of one region on its neighbors. P2 places 16 buffered routers diagonally, dividing the network into 4 equal sized parts. P4 uses 24 buffered routers to divide the network into two isolated circular regions. In P5, 28 buffered routers are used to build a grid to divide the network into 9 small regions with 4 bufferless routers. In addition, we also studied the placement of P6 where each bufferless router is surrounded by 4 buffered routers. Finally, we studied the two extreme cases of router placements: all routers are bufferless (P0) and all routers are buffered (P8).

Figure 4.4, Figure 4.5, Figure 4.8 and Figure 4.9 illustrate respectively, the performance, power and area overheads of each of the placements shown in Figure 4.3. We can observe from Figure 4.4 that NoCs with buffered routers have better performance than NoCs with only bufferless routers. Especially the two end cases of the continuum: BLESS NoC (p0) and VC NoC (vc), where NoCs with virtual channels can improve performance by 30% compared to the BLESS architecture (mix-1 and mix-3). The heterogeneous NoCs improve performance from 4% (p1) to 15% (p8). However, performance is not solely decided by the number of buffered routers. In other words, more buffered routers do not neces-
Figure 4.4: Normalized performance of different router placements.

Figure 4.5: Normalized power of different router placements.

Figure 4.6: Performance of router placements with different buffer size.

Figure 4.7: Power of router placements with different buffer size.

sarily bring better performance. For example, Placements 5, 6 and 7 have 28, 32 and 36 buffered routers, respectively. The performance of placement 5 is almost as good as that of the other two. From the above analysis, we can draw the conclusion that the location of buffered routers in our heterogeneous network plays an essential role in the overall performance. Intelligent placements can reduce the
data latency more effectively under the same resource constraints. This observation implies that, by cleverly placing buffered routers, one can achieve a better tradeoff between performance and cost.

Figure 4.5 illustrates the power consumption incurred by each of our placements. Under high network loads, flits tend to be deflected more frequently and it takes them longer to reach their destination. The power consumed by these deflected flits exceeds the power saved from removed buffer writes in the bufferless routers, resulting in more power overhead in bufferless routers. Note that, although the placement P8 does not improve performance significantly over placement P5 by employing more buffered routers, it can decrease the power consumption by about 10%. This implies that the design space can be explored in multiple dimensions: designers can select an optimal placement to meet the performance and power requirements at the same time.

Figure 4.8 and Figure 4.9 illustrates the area saved by each of our placements compared to the baseline VC architecture. We see that our heterogeneous NoC design still keeps the advantage of bufferless NoCs in saving area: significant amount of buffer area can be saved compared to VC based schemes. In our configurations, adding buffers to 38% of the NoC routers (24 buffered routers in placement 4) only decrease the buffer savings by about 10%. When we consider the overall area of the NoC, most of our heterogeneous NoCs have an overhead less than 10% over
BLESS. The worst case is when all the routers are equipped with buffers, which incurs the area overhead of 18%. One possible concern about heterogeneous NoC architecture is that CMP nodes usually are designed in grids. Employing routers with different structures leads to unbalanced grid sizes. For example, the space for bufferless routers in a grid may not be fully utilized if the unit grid size is chosen uniformly as that of the buffered routers which is larger. This problem can be solved by combining several neighboring heterogeneous core/router pairs into one basic unit. For example, in P6, 4 neighboring nodes (that forms a square) can be selected as the basic building unit for a grid. Since our heterogeneous placements are symmetric, it is not hard to build such grid units from neighboring nodes.

To fully explore the design space, we also experimented with different buffer sizes in our buffered routers as shown in Figure 4.6, Figure 4.7. Generally the data delay and power consumption increase as buffer size decreases because more deflected packets increased network congestion and power.

After exploring the design space of heterogeneous router placements, we next investigate two complementing techniques that take advantage of this heterogeneity: buffered router aware application thread mapping and routing. To support different service classes of applications running on a NoC, we can apply the our BRAM technique to map applications of higher priority close to buffered routers and use the BRAR routing algorithm to route data of such applications through the path near those buffered routers.

### 4.3.2 Heterogeneity Aware Application Mapping

In our heterogeneous NoC, since routers have different capabilities when transmitting data flits, mapping applications to specific regions in the network could result in a better performance if heterogeneity could be taken into account. Based on this observation, we propose a buffered router aware mapping (BRAM) strategy to map application threads as close as possible to buffered routers. Figure 4.10 illustrates two mapping strategies for an application that runs on 4 cores: mapping 1 (vertically hashed cores) and mapping 2 (horizontally hashed cores). Mapping 1 has two neighboring nodes that have buffered routers, while mapping 2 has eight. Since buffered routers can potentially reduce transmission latency by keeping data
closely to the shortest path, mapping 2 of the application will have better chance to reduce the delay. Figure 4.11 illustrates how to apply our BRAM technique to a different placement plan (diagonal). To facilitate the BRAM mapping strategy, we designed an algorithm (Algorithm 1) to calculate the proximity information of each router toward neighboring buffered routers. This algorithm takes, as input, a bitmap indicating the router placement of a heterogeneous NoC. There are two categories of outputs per router node: a distance vector and a weight. Distance vector indicates the number of hops from one router toward a nearest buffered router in each of the four directions. If, in one of the directions, there is no buffered router between a router and the edge of the NoC, we set the distance vector value of this direction to be the size of the NoC’s X (or Y) dimension. The weight on the other hand is the sum of distance vectors in all directions. Our BRAM mapping is quite straightforward so it is not shown here: when mapping an application to a core, always select the one with the lowest weight, in order to utilize the buffer resources.

4.3.3 Heterogeneity Aware Routing Algorithm

With routers having different abilities to move data along the shortest path, another knob that can be used to improve performance is data routing. We propose a BRAR routing algorithm that tries to move data flits along the buffered routers. We use the distance vectors calculated from Algorithm 1 to choose an optimal routing output port. At each node, when the routing logic tries to decide the next
Algorithm 1 Algorithm to calculate buffered router proximity

**Input:** Bitmap of router placement on a heterogeneous NoC.

**Output:** For each router node in the NoC, this algorithm gives: (1) the distance from the router to a closest buffered router in all directions as Distance[East, West, North, South]. (2) Weight of each node indicating the overall closeness of this node to neighboring buffered routers.

\[N \leftarrow \text{all nodes in the NoC}\]

\[Max_{\text{hop}} \leftarrow \text{Max(size of dimension X, size of dimension Y)}\] of the NoC.

```
for i in N do
    Weight_i \leftarrow 0
    for dir in (East, West, North, South) do
        Hop_cnt \leftarrow 0
        Current_node \leftarrow i
        Next_node \leftarrow \text{Neighbor of Current_node in Direction dir}
        while Next_node \neq \text{Nil} \land Next_node \text{ is not a buffered router} do
            Current_node \leftarrow Next_node
            Hop_cnt ++
            Next_node \leftarrow \text{Neighbor of Current_node in Direction dir}
        end while
        if Next_node is a buffered router then
            Hop_cnt ++
            Distance[dir] \leftarrow Hop_cnt
            Weight_i \leftarrow Weight_i + Hop_cnt
        else
            Distance[dir] \leftarrow Max_{\text{hop}}
            Weight_i \leftarrow Weight_i + Max_{\text{hop}}
        end if
    end for
end for
```

node to send a data flit, it checks the Distance vector of current router to find out which direction can reach a buffered router in the shortest distance. Algorithm 2 describes our BRAR algorithm, and Figure 4.12 and Figure 4.13 illustrate how our proposed routing strategy work with grid and diagonal placements. For example, in Figure 4.13, the source node can send flits to either North or East ports. However, the Distance vector of North is 3 which is larger than the Distance vector of East (2 in this case). Our BRAR algorithm consequently chooses East to send flits. The Distance vectors for each router are stored in the router locally, with a storage overhead of a few bytes which is negligible. Note that our BRAR algorithm is not
Algorithm 2 Buffered Router Aware Routing Algorithm

**Input:** Current node, Destination node, Distance vector of Current node.

**Output:** An output port desired to send the data out.

1. $Optimal_X \leftarrow$ the X direction from X-Y routing using Current and Destination nodes information
2. $Optimal_Y \leftarrow$ the Y direction from Y-X routing using Current and Destination nodes information
3. $Hop\_cnt_x \leftarrow$ Distance[$Optimal_X$]
4. $Hop\_cnt_y \leftarrow$ Distance[$Optimal_Y$]
5. if $Hop\_cnt_y \geq Hop\_cnt_x$ then
   - select $Optimal_X$ direction
else
   - select $Optimal_Y$ direction
end if

expected to increase the length of a router’s critical path since the extra steps are only lookup and comparison of Distance vectors.

4.4 Experimental Setup

We assume an 8X8 CMP connected by a mesh network at 45 nm technology. The memory hierarchy is implemented through a two-level directory cache coherence protocol. Each core has a private L1 cache and the L2 cache is shared among
Processors | SPARC 2 GHz processor, two-way out of order, 64-entry instruction window
---|---
L1 Cache | 64 KB private cache, 4-way set associative, 128B block size, 2-cycle latency, split I/D caches
L2 Cache | shared L2 cache, SNUCA with 1MB banks per core, 16-way set associative, 128B block size, 6-cycles latency, 32 MSHRs
Memory | 4GB, 260 cycle off-chip access latency
NoC | 2-stage pipeline, 128 bit flits, 4 flits per packet, Hot Potato routing

Table 4.1: Baseline configuration.

<table>
<thead>
<tr>
<th>App 1</th>
<th>App 2</th>
<th>App 3</th>
<th>App 4</th>
<th>App 5</th>
<th>App 6</th>
<th>App 7</th>
<th>App 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>apache apsi</td>
<td>barnes ocean</td>
<td>sjbb swim</td>
<td>apsi ocean</td>
<td>apache swim</td>
<td>sjbb apsi</td>
<td>apsi barnes</td>
<td>ocean swim</td>
</tr>
</tbody>
</table>

Table 4.2: Applications formed with a pair of OpenMP multi-thread benchmarks.

all cores. The detailed configuration is described in Table 4.1. We evaluate the performance and power consumption of our proposed heterogeneous NoCs using a cycle-accurate NoC simulator [118]. We use Orion [113] to evaluate our design’s cost in terms of buffer area. Each router has 5 input ports and 5 output ports. The router pipeline latency is 2 cycles, and it takes 1 cycle to traverse a 128-bit wide link. We assume each data packet contains 4 flits and each flit has 128 bits. In the baseline virtual channel based configuration, we assume there are 4 virtual channels in each router and that the virtual channel is 8 flits deep. Our heterogeneous NoC is configured as follows: input buffers are completely removed from our bufferless routers, and there are input buffers of 8 flits deep in each input channel of our buffered routers. We used the SPEC OpenMP [114] benchmarks as our application programs. In our settings, each application is run with 16 threads.

<table>
<thead>
<tr>
<th>Mix 1</th>
<th>Mix 2</th>
<th>Mix 3</th>
<th>Mix 4</th>
<th>Mix 5</th>
<th>Mix 6</th>
<th>Mix 7</th>
<th>Mix 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>App 1, 2, 3, 4</td>
<td>App 1, 3, 4, 5</td>
<td>App 2, 3, 4, 5</td>
<td>App 2, 4, 5, 6</td>
<td>App 3, 4, 5, 6</td>
<td>App 3, 5, 6, 7</td>
<td>App 4, 5, 6, 7</td>
<td>App 5, 6, 7, 8</td>
</tr>
<tr>
<td>4.71%</td>
<td>5.17%</td>
<td>4.95%</td>
<td>7.15%</td>
<td>5.7%</td>
<td>5.8%</td>
<td>7.58%</td>
<td>4.18%</td>
</tr>
</tbody>
</table>

Table 4.3: Mix of applications mapped to a 8x8 NoC.
and we collect memory access traces of these application threads. To stress the network with high traffic loads, we map two threads from two applications to each core. Table 4.2 lists our applications that are mapped to CMP cores. We created 8 different mixes in our evaluation which are described in Table 4.3. Each Mix contains 64 threads from 4 applications running with 16 threads each. Injection rate (message/cycle/node) is also shown. We compare our results with two baseline configurations: NoC with bufferless routers (called P0) and NoC with buffered routers using virtual channel for control flow (VC).

### 4.5 Results

Figure 4.14 and Figure 4.15 show our evaluation results when BRAM and BRAR are applied to diagonal and grid placements (i.e., P2 and P5). In the diagonal placement, performance improvement using BRAM can be as high as 18%, while in the grid placement, the maximum performance improvement is around 9%.
The reason why the BRAM works better in the diagonal placement case is that it can fully utilize buffered routers. Without BRAM, the performance of BLESS and that of our heterogeneous NoC would be very close to each other, due to the fact that the buffered routers are separated by more bufferless routers, so the deflection within each region surrounded by buffered routers cannot be effectively controlled. BRAM maps applications intentionally close to those buffered routers, fully utilizing the ability of these buffers to reduce the flit latency. In contrast, in the grid placement, without BRAM, the heterogeneous NoC already improves the performance due to the fact that more buffered routers are placed in the network, and deflections are already reduced by those routers even without using BRAM.

The results plotted in Figure 4.15 show BRAR alone can reduce the delay cycles up to 15% in the diagonal placement. However, BRAR alone does not improve the performance significantly when used on the grid placement. This is because the buffered routers in the grid placement are connected in X and Y directions and the X/Y routing algorithm send all flits to the buffered routers. As a consequence, congestions occur in the buffered routers, leading to longer delays. This problem does not happen in the diagonal placement, because the X/Y routing diverts flits to both X and Y direction before finding the next buffered router, so congestion
in buffered routers is greatly reduced.

Finally, we evaluated performance when both techniques are applied in our experiments. The performance of our heterogeneous NoC employing these two techniques comes very close to the virtual-channel based NoCs. Most of our heterogeneous NoCs can achieve an average flit delay within 5% range, compared to VC based NoCs. Further, our design significantly reduces the buffer area costs. To investigate the scalability of our proposed techniques, we experimented with a larger NoC network of 100 nodes as shown in Figure 4.16 and Figure 4.17. The result shows our scheme works with large scale NoC as well.

4.6 Conclusion

Aggressive many-core designs involving dozens or hundreds of processing cores put a limit on the resources that can be accommodated in on-chip networks. Several bufferless NoC designs have been proposed to reduce the cost in area and power, such as BLESS [107], BPS [111] and SCRAB [108]. Removing buffers from all of the NoC routers significantly reduces the area cost but cannot deliver satisfying performance in all cases. In this work, we propose a low cost heterogeneous NoC architecture using both buffered routers and bufferless routers. To extensively explore the design space, we evaluated several router placement plans with different number of buffered routers and positions. Our simulation results show that intelligent router placement can achieve significant gains in performance under the same resource budget. We also propose two techniques that take advantage of the NoC’s heterogeneous nature in order to further improve performance. BRAM is an application mapping technique that can map applications close to buffered routers to improve performance and BRAR is a routing algorithm that sends data packets along the buffered routers to reduce chances of misroutings. Our simulations show that applying such techniques can reduce data latencies by an average of 15%. In summary, heterogeneous NoC architectures incorporating both buffered and bufferless routers open up a new design space for performance improvement and cost reduction. Our design and evaluation demonstrate that significant potential benefits can be possibly explored in this area.
A new challenge in CMP designs is the management of dark silicon. As a major power management technique, DVFS has increased design complexity and cost as multicore systems scale. In this chapter, we propose to design an effective yet simple dynamic component power re-allocation scheme in the presence of power emergency. Among the on-chip components, the cores and caches consume most of the power. Gating off these two types of components can significantly reduce power consumption. However, the chip’s performance is largely dependent on these two resources. This brings the challenge of how to effectively utilize these resources in order to achieve the best performance in the face of power emergencies.

We observe that programs exhibit different scalability characteristics with respect to the number of cores and the size of caches. Running programs with more cores or larger caches does not always bring better performance. Based on these observations, we propose a dynamic scheme called TaPeR, that can (1) predict the scalability of parallel programs with respect to core count and cache capacity; (2) selectively gating off cores or caches in order to satisfy the power constraints; and (3) achieve optimal performance at the same time.

The design proposed in this chapter targets on improving the power-efficiency as shown in the shaded part of Figure5.1.
Figure 5.1: Interrelationship between performance, power and reliability characteristics of many-core multiprocessors.

5.1 Introduction

The percentage of a chip that can be actively used while remaining within the power budget is continuously dropping. This is termed as utilization wall [124] and leads to a portion of the chip - called dark silicon [125] - remaining unpowered in emerging multicore architectures. Recent works from both academia and industry [126, 127, 128, 129] discuss different strategies where dark silicon is exploited.

Dynamic voltage and frequency scaling (DVFS) is a widely-studied power management scheme that dynamically adjusts voltage and frequency levels based on the run-time needs of an application. However, conventional DVFS techniques may miss the fine-grained behavioral changes of applications because they rely on samplers in the OS to adjust their policies. DVFS also faces the limitation of high costs in future many-core systems.

As a result, several techniques have been proposed to overcome the power challenges other than DVFS [130, 131, 132, 133, 138]. One common characteristic of these works is that they exclusively focus on cores in the architecture, implicitly assuming that the biggest power-performance tradeoffs can be achieved by
reducing power consumed by cores, largely ignoring the uncore components. Our experimental data shows that, depending on the configuration of the target multicore chip, uncore components (especially, large last level caches (LLCs)) can also contribute to a significant fraction of overall power consumption during execution. Further, under the same power budget, the best power re-allocation strategy (i.e., whether some cores, some caches or a combination of both needs to be gated off) depends on application characteristics. Therefore, always decrease power allocated to cores without considering applications’ sensitivity to cores and caches may not be the best option from a performance perspective.

Motivated by these observations, we propose a dynamic scheme for multi-threaded applications to react to power emergencies on a many-core CMP system. Our goal is to flexibly map the applications to on-chip resources in order to make the most out of the power available. The key to achieving this is to allocate resources by taking the scalability of each application into account. Our proposed scheme is based on the observation that applications show varying scalability to resources such as cores and LLC. For example, some applications perform better when running on more cores while other applications achieve higher performance with larger caches. With different power budgets or execution phases, an application can exhibit different scalability/sensitivity. By taking advantage of such characteristics, our scheme can achieve better performance than simply gating off cores or caches when a power emergency strikes.

Our evaluations reveal that:

- Under the same power budget, some applications prefer running with more cores whereas some other applications prefer larger caches, for the best performance. Also, for some applications, the best performance is achieved by a coordinated power re-allocation to both cores and caches.

- When the available power budget is changed, the best power re-allocation strategy for an application also changes in some cases.

- Some applications exhibit a phase based behavior, and consequently prefer different power re-allocation strategies (core vs. cache) even if the available power budget is kept constant during execution.
• A dynamic strategy is the best option when the power budget modulates during the course of execution.

This work makes the following main contributions:

• We propose an architectural level strategy, called TaPER, that decides the best components to be allocated with more power considering the available power budget. This scheme can have both static and dynamic versions. The static version makes a power re-allocation decision at runtime but does not revisit that decision during the remaining part of execution. In contrast, the dynamic version considers the phase behavior of applications and can change the power re-allocation strategy periodically.

• We experimentally evaluate the proposed strategy using a set of multi-threaded programs drawn from different benchmark suites. The results show that we can achieve about 21% performance improvement compared to simply gating off cores/caches under the same power budget.

5.2 Motivation

5.2.1 On chip power breakdown

Due to power emergencies, a fraction of the chip may have to be turned off in the dark silicon era. In this work, we target a scenario of power emergencies, as a result of which, the power supply to some on-chip components has to be gated when applications are still in the middle of their execution. To simplify the chip design, instead of applying DVFS techniques, we focus on designing a power re-allocation strategy that can achieve optimal performance while keeping the power consumption within a specified bound (called power constraint or power budget in this work).

We start our discussion by studying the breakdown of chip power in modern multicore systems. The top ones on the list are cores because they are usually the most power hungry. However, as the technology nodes keep shrinking, the percentage of power consumed by on-chip caches increases significantly and can become comparable to the power consumed by the cores. Figure 5.2 shows the
breakdown of power consumption between a core and a cache under different processing technologies. We used CACTI [148] and McPat [147] to collect the data shown. The cores being investigated consist of four-way out of order CPUs with 64-entry instruction window. Inside each core, there are split instruction/data L1 caches. The L1 caches are 4-way set associative and employ a write-back policy. The L2 caches studied here are 16-way set associative with 64B block size. We assume the cores run at 2 GHz and the L2 cache latency is 20 cycles. As shown in the figure, with 90 nm technology, the power of a 1MB cache is only about 12% of that consumed by a core. The cache power is so small that it is negligible compared with the core power. However, the percentage of cache power increases greatly with the advance of processing technology. When the technology node shrinks to 22 nm, a cache of 512 KB consumes almost the same amount of power as a core. This observation means that caches should be considered along with cores in designing a scheme that can respond to power emergencies.

Figure 5.2: Power breakdown between cores and caches (L2) under different technology nodes. The size of L2 caches ranges from 64 KB up to 1MB.

5.2.2 Target CMP architecture and power re-allocation schemes

Figure 5.3(a) shows an example of the tiled CMP architecture studied in this work. There are 8 tiles, each consisting a core and an L2 cache. Inside each core resides a CPU and its instruction and data L1 caches. For this study, we configured our LLC as private L2 caches after balancing the design trade-offs between private and shared L2 caches. The reason is the private caches scale better to a tiled CMP with large number of cores compared to shared caches [134, 135]. As discussed in the previous section, both cores and caches need to be considered in designing
Figure 5.3: The CMP architecture and power re-allocation scenarios due to power emergency investigated in this work. Cores or caches that are gated off are represented by dark regions. Scenario (a) represents a baseline case where the CMP is running with full power budget. Scenarios (b)-(d) illustrate possible power re-allocation methods that satisfy a same reduced power budget.

a "performance-aware" power re-allocation scheme. We assume that when a core is gated off, the task running on this core is moved to other running cores. The decision of how tasks are migrated is made by the OS. We also assume that part of the caches can be powered off. In fact, there is a rich supply of works investigating various schemes to power off parts of a cache to save power [140, 141, 142, 143, 144]. In order to simplify the hardware design, we assume that all the L2 caches powered on are configured with the same size. For example, if we want to reduce the power of a 1MB cache by half, then the new L2 cache size change to 512KB for all private L2 caches. In addition, the cache size is always reduced by the order of 2 under our power re-allocation strategy. If the fully configured L2 is 1MB, then the possible L2 cache sizes after the power re-allocation are 512KB, 256KB, 128KB and 64KB. To reduce the design cost, a cache is not reconfigured in our study to be use by cores of other tiles when its core is no longer running a program. In other words, a private L2 cache or part of it is turned on only when its core is powered on.
Figure 5.4: Characterization of parallel applications showing different scalabilities. The execution time of each application is normalized with respect to that of the full configuration (48 cores and 1MB private L2 cache). Each application is run with 48 threads.
Based on the above assumptions, there are several possible power re-allocation methods as shown in Figure 5.3(b)-(d) that all satisfy a given power budget. The most intuitive power re-allocation method is the one shown in Figure 5.3(b). When we gate off a core, we remove the whole tile including its private L2 from the running configuration at the same time. Tasks originally running on core 4-7 need to be moved to cores 0-3. In this scheme, power is saved by cutting down power consumption of whole tiles. The remaining tiles are still fully powered on but may have more threads running on them depending on where the tasks are migrated to. A second power re-allocation method is shown in Figure 5.3(c), where the number of cores remains the same. Instead, each core’s L2 size gets reduced to a quarter of its original. No thread re-mapping is needed in this way. Figure 5.3(d) shows another method where not only are some tiles gated off, also the sizes of L2 caches in the remaining tiles are reduced. Compared with Figure 5.3(c) and (d), this method has more cores than the first method and larger L2s than the second method.

The problem we would like to address in this work is which method of power re-allocation needs to be used to (i) remain under the specified power constraint, and to (ii) achieve the best performance under that power constraint. Obviously the effectiveness of these methods depends on the application’s scalability: if an application’s performance scales with the number of cores more than with L2 sizes, method (c) is the best choice. If on the other hand, an application scales with L2 size mostly, method(b) will certainly perform better.

We used several parallel programs to investigate these two scalability characteristics: the scalability with the number of cores and the scalability with the size (capacity) of L2 caches. Results are shown in Figure 5.4.

For benchmark mgrid, the performance does not scale with the number of cores. One can observe that the execution time remains almost the same when we reduce the number of cores from 48 to 18. However, the performance degrades significantly with smaller L2 sizes. For example, with 48 cores running, the execution time doubles if we reduce the L2 size from 1 MB to 64 KB. Obviously this type of application benefits most from the power re-allocation method shown in Figure 5.3(b). In contrast, fluidanimate scales with the number of cores rather than with cache sizes. Reducing the number of cores cuts the performance by half, while
running with smaller L2 sizes lead to little performance change. For this type of applications, we should choose the power re-allocation method in Figure 5.3(c). By analyzing Figure 5.4(c), we find that the performance of bodytrack is affected by both core counts and L2 sizes. Therefore the scalability of bodytrack cannot be categorized into either of the first two groups. For this application, running on 48 cores with 128KB has similar performance with running on 18 cores and 1MB L2. For this type of application, simply reducing the number of cores or the size of L2 caches does not bring optimal performance. Instead, a power re-allocation scheme similar to Figure 5.3(d) will achieve better performance. The next application we studied is apsi which shows different scalability in two regions as shown in Figure 5.4(d). When the number of cores is greater than 38, it scales with L2 sizes. Otherwise, this application scales with number of cores. This implies that the best power re-allocation scheme suitable for this type of applications depends on the power available. If the power reduction is not significant and the running cores are above a certain amount, reducing the number of cores will lead to a better performance. On the other hand, when there is not sufficient power to turn on enough cores, we should reduce the cache sizes in order to get better performance.

From the above analysis, one can draw the conclusion that there is not a single power re-allocation scheme that fits for all types of applications. That is, for some applications, one should perform fewer cores and for others smaller caches in order to achieve the best performance. This is the main motivation behind TaPER, a scheme that can flexibly make power re-allocation decisions based on an application’s scalability/sensitivity with respect to cores and caches.

5.3 TaPER: Our Proposed Power Re-allocation Strategy

We designed TaPER, a scheme that adaptively allocate power to CMP cores and (or) caches based on the program sensitivity to these components in order to achieve high performance in the presence of power emergencies. TaPER dynamically predicts the scalability characteristics of multi-threaded applications through a curve fitting method. The key idea behind this scheme is that with the con-
straints of power consumptions, the best configuration of core and caches to power on depends on the scalability of programs which changes with individual application, the power budget available and even application execution phases. TaPEr attempts to capture the program dynamics and make the best use of the power available to achieve optimal performance. This section provides the details for the design of TaPEr as well as the scalability prediction method.

5.3.1 Measurement of performance

To select an ideal configuration, we first need a metric to evaluate the performance. There have been several metrics used to measure the performance of parallel applications, such as Instructions per Cycle (IPC), weighted speedup, execution time and Average Turnaround Time (ATT). Among these, the weighted speedup and ATT are more suitable for multiple multi-threaded applications which are not the focus of this work. Consequently, we focused on the remaining metrics. The best metric of a program’s performance is the total time used to complete the execution of the program. However, it is not possible to be used to measure the performance in the middle of execution. We decided to use a metric called TPCI (time per committed instructions) to measure the performance. The TPCI values can be easily obtained from hardware performance counters. A metric similar to TCPI is used in [136] which uses time per executed instructions to measure performance. We choose committed instructions instead of executed instructions in TPCI because they can reflect the performance more accurately by removing the factors such as mis-predicted branches.

5.3.2 Predicting scalability

When a power emergency occurs, TaPEr needs to make power re-allocation decisions based on the application’s scalability. It is not practical to sample all possible configurations to collect scalability information. Motivated by this, we perform sampling on a few configurations during the execution and apply curve fitting techniques to predict the scalability for all configurations. In our scheme, a scalability table is used to store performance information for all configurations. The table is indexed by two keys: (core count, L2 cache size). Given these two
keys, we can retrieve the predicted performance of a certain configuration. By analyzing the performance of all configurations, our scheme can figure out how the program scales with cores or caches. Then an appropriate configuration is selected to continue executing the program.

There are various methods to predict the scalability. In this work, we implement our scalability predictor using a curve fitting method. We select linear interpolation as our curve fitting model because this method has relatively high accuracy and low cost which fits into our on-line approach. Figure 5.5 shows the results of the curve fitting methods. Figure 5.5(a) plots the performances using 60 sampling points while Figure 5.5(b) shows the results based on only 12 samples. We can see that the curve fitting method can accurately capture the performance trends with

![Figure 5.5: Plots of real and predicted performance for *apsi*.
only a small number of samples. We propose to use a single thread running on one of the cores to collect the samples and running the curve fitting algorithm.

Next, we need to evaluate the cost of scalability prediction in TaPEr. There is extra power overhead to predict the scalability in our scheme such as collecting samples and run the curve fitting algorithm. However, compared with the power consumed on the application threads, this extra power consumption caused by TaPEr is relatively low since the curve fitting algorithm is not called unless there is a triggering event such as a new power emergency. Overall, compared to the potential benefits brought by TaPEr, its overhead is negligible because more power will be consumed in the long run if the application executes on a non-optimal configuration.

5.3.3 Static TaPEr vs. dynamic TaPEr

Suppose that, at the time of a power emergency, we have collected the statistics that can help us to predict the program scalability under the new power budget. Based on such information, we can make a decision about the new configuration that will have the best performance. In the case of static TaPEr, we do not change the decision during the program’s execution. However, we found such a static TaPEr scheme is not optimal because some programs have changing phases which displays different scalabilities.

For example, canneal experience three phases during its execution as shown in Figure 5.6(a). The program first scales with cache sizes. Then the program changes to scale with number of cores in the second phase. Finally, the program changes back to scale with cache sizes in the third phase. Figure 5.6(b) illustrates the relative performance and length of each phase. For such applications, if we apply the static TaPEr scheme, a configuration with a small number of cores but large L2 sizes will be selected for the program because it is the best configuration for the first phase. However, as the program enters the next phase, such a configuration will result in severe performance degradation. This motivated us to design a dynamic TaPEr scheme that can capture the changing behavior of different phases and adjust its decision accordingly.

In dynamic TaPEr, the TPCI is measured every epoch (or time window). Whenever we observe a change of the performance counter values greater than
a certain threshold, the dynamic TaPEr scheme is triggered to start sampling and predicting the new scalability. Based on these results, a new decision is made that fits best for this new phase.

5.4 Experimental Evaluation

5.4.1 Experimental setup

We evaluated our proposed TaPEr scheme using the Gem5 simulator [146]. Our target ISA is ALPHA and we run the simulations in full system mode. Our simulated multicore system is built with out-of-order cores running at 2GHz. The CMP system we modeled consists of 48 cores. Each core has split instruction and data L1 caches. The on-chip last level caches are private L2 caches with a maximum capacity of 1 MB. When the chip works under full configuration, all of the 48 cores and their L2 caches are turned on. The power consumed in this mode defines the
upper bound of the chip’s power consumption. When the chip encounters a power emergency, the power consumption has to be cut down from the full configuration state. We assume the system has the ability to gate off either cores or part of the L2 caches (existing strategies such as [141] and [143] can be used for that). L2 caches can be gated off by the power of 2. So the available L2 capacity is 1MB, 512 KB and down to 64KB. To simplify the cache design, all the private L2 caches will be configured to a same size if a decision is made to gate off part of the L2 caches. We use CACTI [148] and McPat [147] to compute the power consumptions of L2 caches and cores. Table 5.1 provides our major simulation parameters and their default values.

We choose PARSEC [149] and SPECOMP [150] as the workloads to evaluate our proposed TaPEr scheme. For PARSEC programs, we collect simulation statics after the programs enter the parallel region marked by ROI. We instrument the SPECOMP programs with checkpoints so that we can study the parallel executing code regions. We select the sim-small input sets for the PARSEC benchmarks and run them to completion to get the characteristics. For the SPECOMP benchmarks, medium-sized inputs are used and the characteristics are collected on representative parallel regions due to the large size of the benchmarks. Each program is run as 48 parallel threads. As a result, if there is no power emergency, each core is running one thread of the application. If some cores need to be gated off, the threads running on these cores are migrated to the remaining cores by the OS. We do not consider re-threading of applications in this study.

<table>
<thead>
<tr>
<th>Processors</th>
<th>ALPHA ISA processor, 2 GHz frequency, four-way out of order, 64-entry instruction window</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 Cache</td>
<td>32KB/core, private, 4-way set associative, 64-byte block, 3-cycle latency, write-back, split instruction/data caches, 32-entry MSHR</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>private L2 cache, 16-way set associative, 64B block size, 20-cycles latency, 22 MSHRs. Full L2 configuration is 1MB, with other possible L2 sizes of 512KB, 256KB, 128KB and 64KB.</td>
</tr>
<tr>
<td>Memory</td>
<td>2GB, 260 cycle off–chip access latency</td>
</tr>
<tr>
<td>Epoch Interval</td>
<td>50 million instructions</td>
</tr>
</tbody>
</table>

Table 5.1: Baseline CMP configuration.
5.4.2 Evaluation results

We implemented two types of our proposed TaPEr schemes: static TaPEr (s-TaPEr) and dynamic TaPEr (d-TaPEr). We compare their performances against several alternate power re-allocation strategies: cache favoring (as illustrated in Figure 5.3(b)) and core favoring (as illustrated in Figure 5.3(c)). Two optimal schemes are also used in the comparison. The static optimal scheme (s-opt) does not consider program phase changes and the dynamic-optimal scheme (d-opt) selects optimal configurations for different program phases. We first evaluate all six schemes using three case studies by studying how workloads react to dynamically changing environment. Next, we present the aggregate results over all of the 11 workloads, showing that the benefits of our scheme hold over different categories of workloads. Finally, we examine the effect of TaPEr on systems with shared L2 caches as a sensitivity analysis.

5.4.2.1 Case study I: different workloads under the same power budget

We use this case to investigate how different workloads benefit from power re-allocation strategies. To make the result clear, we use static schemes in this case study and use other case studies to investigate dynamic schemes. We evaluate different schemes with workloads mgrid and fluidanimate in this case. Figure 5.7 illustrates the performance of these two workloads normalized to that of the optimal configuration. The workload fluidanimate benefits more from core favoring scheme (allocating more power to cores) because its performance scales with the number of cores. With the two tested power budgets, core favoring scheme achieves about 20% more performance than cache favoring scheme. By contrast, mgrid scales with cache size because it has a larger working set. Shutting down L2 caches results in performance degradation of 18% when the power budget is 80% of the full configuration. When the power budget further reduced to 60%, the performance of core favoring scheme degraded to 51% of cache favoring scheme. In contrast, the static TaPEr scheme achieves near optimal performances for both workloads. These results validate our purpose of designing the TaPEr scheme: simply favoring cores or caches does not bring the best performance for all workloads. A flexible
scheme such as the TaPEr is needed to select the best configurations based on the special characteristics of the workloads.

![Performance diagram](image)

**Figure 5.7**: Evaluation of schemes on workloads with different scalabilities.

### 5.4.2.2 Case study II: same workload with different power budgets

The goal of this case study is to measure how different schemes perform under different power budgets. As plotted in Figure 5.4(d), whether apsi scales with core count or L2 capacity depends on the power available. When the available power budget is not tight, a larger L2 cache leads to better performance. If the power gets further reduced, the workload’s performance will be decided by the number of cores. In our experiment, we simulate the power changes on a fixed execution phase in order to clearly show the reactions of each scheme. Initially, the power is cut down to 90% of the full configuration where all cores and L2 caches are powered on. Then it is further reduced to 80%, 70% and 60%. After that, we assume the power emergency is relieved and the power budget gradually increases back to 90%.

As shown in Figure 5.8, when the power budget is 90% of the original, the program scales with L2 size because there are enough cores to run its threads. As a result, the cache favoring scheme is the optimal choice. However, when we reduce the power further down to 60%, reducing the power consumed by caches becomes a better choice. This is because there is not enough power to turn on more than 38 cores and the performance of the program degrades more with reduced number of cores. As can be observed from Figure 5.8, core favoring with 60% power has a performance 31% better than cache favoring scheme. When the power budget increases back to 90%, we can see that the workload again changes its scalability,
favoring the cache favoring scheme eventually. As our results clearly indicate, neither cache favoring or core favoring uniformly gives good results under different power budgets. Only TaPEr can achieve a good performance comparable to the optimal scheme because it can adjust its decision based on power budget changes. We also observe that the performance of TaPEr is sometimes better than the cache favoring or core favoring scheme, such as with 60% power budget. This is similar to the case in Figure 5.3(d) where power is distributed between caches and cores with balance so that higher performance can be achieved.

![Figure 5.8: Evaluation of schemes with changing power budgets using apsi where power budget changes between 90% and 60% of the full configuration.](image)

### 5.4.2.3 Case study III: same workload with different program phases

As we have shown in Figure 5.6, there are three phases in the execution of application `canneal`. In this experiment, we simulate the scenario when the power budget has to be cut down to its 60%. Under the simple core/cache favoring schemes, both 28 cores running with 1MB L2 and 48 cores with 64K L2 have a power consumption within this limit. The performance scales with L2 sizes but not the number of cores in this phase. As a result, configurations with fewer cores but larger L2 cache have better performance. Figure 5.9 shows that cache favoring can achieve the same performance as the two optimal schemes. Shutting down caches has the lowest performance. In the second phase, the scalability of the application changes dramatically. Reducing the amount of cores leads to more performance degradation compared to reducing the L2 capacity. However, the static schemes cannot capture such dynamic changes. s-opt and s-TaPEr stick to their decisions based on the behavior of the former phase. As a result, they have performance degradation of 13% and 22% respectively. In contrast, the d-TaPEr scheme adapts
its decision based on the changed scalability, achieving a performance as good as 97% of the dynamic optimal scheme. As the program enters the third phase, it once again scales with the number of cores. In this case, running with 28 cores with full L2 size again becomes the best configuration. The d-TaPER adjusts its configuration for the second time and keeps a near-optimal performance. Note that here d-TaPER has a performance slightly lower than the best static schemes which is resulted from the inaccuracy of the prediction algorithm.

![Figure 5.9](image)

Figure 5.9: Evaluation of schemes with dynamically changing phases using canneal.

This case study shows that simple core/caches favoring schemes or statically predicting the scalability does not bring the best performance. A dynamic scheme such as d-TaPER is needed in order to adjust the decisions in a timely manner.

### 5.4.2.4 Overall results across all workloads

Figure 5.10 and Figure 5.11 compare all the schemes averaged across the 11 workloads with power budget of 80% and 60% respectively. There are 6 PARSEC workloads and 5 SPECOMP workloads picked to cover the different characteristic categories. We also calculated the harmonic means of all workloads in these two cases. The aggregate results are consistent with the observations made in the three case studies we have shown above. The experiment results show that d-TaPER achieves performance closest to the dynamic optimal scheme in most cases. Compared to simple schemes such as cache or core favoring, d-TaPER can improve performance up to 59% in the best case. On the average, d-TaPER performs 21% better than these simple cache-favoring or core-favoring schemes when
the power get reduced to 60%. d-TaPEr also outperforms s-TaPEr by as much as 16% for benchmarks whose scaling behavior changes with time. Among all the workloads, x264 achieves least improvements using TaPEr schemes. The reason is x264 is benchmark that has very small design space explorable by the TaPEr schemes since it does not scale well whether with the number of cores or L2 cache sizes. Comparing the schemes across different power budgets, our results show that TaPEr schemes outperform the simple cache/core favoring or static schemes significantly.

5.4.2.5 Sensitivity analysis

Besides the private LLC caches we have explored to this point, the LLCs are also commonly designed to be shared among all cores. The scalability characteristics of shared L2 caches may change due to data sharing. In addition, contributions of shared L2 caches to the overall power consumption is different. Therefore, it is important to study the effectiveness of our scheme with shared caches. In our sensitivity study of shared L2 caches, we experimented with varying L2 sizes from 32 MB down to 2MB for SPEC2006 benchmarks. For PARSEC benchmarks, due to their relatively smaller memory footprints, we experimented with the shared L2 cache sizes from 256KB to 4MB. The power budget is set to 80% and 60% of the full configuration. As is shown in Figure 5.12 and Figure 5.13, d-TaPEr scheme
still outperforms other schemes. The average performance of d-TaPEr is 98% of the
dynamic optimal configuration. With 60% power budget, d-TaPEr achieves 96% on
average compared to the optimal scheme. When the available power is reduced to
80%, we observe that most of the PARSEC benchmarks achieve better performance
with the core-favoring scheme. In contrast, most of the SPEC2006 benchmarks
benefit from the cache-favoring scheme. The cause of such a difference lies in the
fact that the PARSEC benchmarks have smaller memory footprints compared to
the SPEC2006 benchmarks. As a result, running the PARSEC benchmarks with
smaller caches but more cores brings more benefit in performance. On the contrary,
the SPEC2006 benchmarks need larger shared caches to maintain relatively high
performances.

Comparing Figure 5.12 and Figure 5.13, we observe that when the power budget
changed from 80% to 60%, most of the benchmarks have better performance with
core-favoring schemes. This is because when the power budget is reduced below
a certain level, the benchmarks require certain amount of parallelism to maintain
higher performance. In other words, when the power budget gets tighter, running
with more threads brings better performance than running with larger caches. This
is because data is shared by multiple cores, thus reducing the need for larger cache
spaces. When we compare the private caches and shared caches results, we observe
that core-faving schemes is more preferred by the shared-cache configurations. The
reason is there is a lot of data sharing in these parallel benchmarks, and the caches
become more power efficient in shared-cache configurations. As a result, allocating more power to cores will more likely bring better performance.

5.5 Related Work

There have been several prior works on scheduling for multi-threaded applications [136, 137, 139]. [136] modified task schedulers in the Operating System to achieve better performance in multicore systems. Similar to our work, their schedulers make decisions based on program scalability. However, there are several differences between [136] and our work. We explore the program scalability on both cores and caches while only scalability with number of cores is considered in [136]. Our goal is to improve performance under power constraints in the dark silicon era. [136] targets on improving the system throughput when several multi-threaded programs are executed at the same time. [137] designed thread schedulers to avoid hardware resource contention. Their optimization target is to reduce the energy delay product which is different from TaPEr. Other than using schedulers in the Operating Systems to re-map threads to cores, there have been works that improve system efficiency through compiler techniques [139, 145]. Based on the estimation of the types of parallel threads and the communication patterns between them, [139] proposed a compilation system that can dynamically adjust the number of threads running together. Thus the technique can improve the program performances as well as increase the utilization rate of a system. The difference between their work and TaPEr is that instead of modifying the compiling system, we use on-line techniques to predict the program scalability and adjust the mapping of threads to cores at run-time. Curve fitting techniques similar to our work is also used in [138] to predict program scalability. However, there are several differences between [138] and TaPEr. Their goal is to reduce the EDP while our goal is to improve performance with a tight power budget. [138] only considered the scalability with number of cores and the number of threads can be changed in the middle of program executions. In our work, we assume the number of threads is not changing and our scheme is based on the scalability with both cores and caches. The past years have also seen many research works leveraging reconfigurable architecture to reduce cache power consumptions [140, 141, 142, 143, 144]. Most of these works
save power by selectively shut down cache ways to reduce the leakage power of caches. TaPEr proposes to use similar techniques to gate off part of the L2 cache. However, in TaPEr, we do not employ fine-grained gateoffs on specific cache ways as done in those works. Instead, we power off a chunk of cache lines together that reduces the design cost.

5.6 Conclusion

In this work, we investigated the schemes of re-allocating power to part of a multicore system when the available power supply is reduced. Even though cores and caches are the major targets to be turned off, simple schemes that reduce the core counts or cache capacity without discretion does not bring the best performance across all types of applications. We have shown that, in order to achieve high performance, scalability of applications to both cores and caches should be taken into account when we make decisions about what resources should be gated off and how much of them should be gated off. We have proposed a technique to dynamically predict the scalability of the applications. We have also proposed a dynamic scheme that can adjust the power allocations based on program phases. We evaluated the proposed scheme and compared its effectiveness with simple schemes as well as optimal schemes. Our results show that our TaPEr scheme can achieve a performance that is very close to the optimal performance in most cases.
Conclusion and Future Work

6.1 Contribution

This dissertation attempts to address some of the reliability and power-efficiency issues in many-core multiprocessor design. The goal is to improve the processor’s reliability, reduce the power/energy consumption, but minimally harm the performance.

Focusing on data reliability, Chapter 2 discusses a control theory based approach designed to improve transient error resilience in shared on-chip caches. The proposed scheme takes two input parameters: performance QoS and reliability QoS. The performance QoS indicates the minimum cache hit rate value acceptable, whereas the reliability QoS represents the desired reliability assurance. With the help of a metric called the Reads-with-Replica (RwR), the proposed technique balances the partitioned cache spaces allocated to data and their replicas. The experiment results show that the proposed scheme is able to satisfy both QoS requirements in most cases, providing both performance and reliability guarantees.

Chapter 3 presents a scheme to improve the reliability of on-chip interconnections. By tuning the error recovery schemes, the scheme dynamically chooses the time for error checking and retransmissions. The technique takes advantage of both end-to-end retransmission and hop-by-hop retransmission. The experimental evaluation shows that the scheme can not only meet the NoC reliability and performance requirements but also reduce power consumption.

Chapter 4 focuses on designing network-on-chips with high performance but
low power consumptions. NoCs evolve from the off-chip networks that have a lot of buffers. However, the power consumed by router buffers and the complexity of managing the buffers becomes costly overheads for on-chip environments. On the other hand, totally removing buffers from NoC routers can lower the cost but lead to significant performance losses. A heterogeneous NoC design is proposed in Chapter 4. The proposed design employs both buffered routers and bufferless routers in a same on-chip network. The bufferless routers can lower the cost of the NoC in terms of power and area while the buffered routers can improve the performance. A Buffered Router Aware Mapping (BRAM) and a Buffered Router Aware Routing (BRAR) algorithm are also proposed to further improve the performance of the heterogeneous NoC design.

Chapter 5 discusses TaPEr, a scheme that balances power consumption between CPU cores and caches for parallel applications to achieve better performance at lower power costs. It is observed that program performances exhibit different scalability characteristics with respect to the number of cores and the size of caches. Running programs with more cores or larger caches does not always bring better performance. Based on these observations, Chapter 5 proposes a dynamic scheme called TaPEr. TaPEr first predicts the scalability of parallel programs with respect to core count and cache capacity. Then it dynamically gates off some cores or caches in order to satisfy the power constraints. By taking advantage of program scalability, TaPEr can achieve near optimal performance with the power available.

6.2 Future Work

The designs and schemes presented in this dissertation may be extended in future works, this section briefly describes their future directions.

6.2.1 Exploiting Heterogeneity in Network-on-Chip Design

As the number of cores in a many-core multiprocessor keeps increasing, the role of NoC becomes more and more important. Bufferless NoC has been proposed as an option of cost-efficient design. With low network traffic, bufferless NoC can achieve comparable performance to buffered NoC but at much lower costs. However, with
the increasing number of cores in many-core multiprocessors, the network traffic tends to increase too. In such situations, bufferless NoC cannot provide desirable performance or power guarantees. In Chapter 4, we proposed a heterogeneous NoC design that selectively adds buffers to the baseline bufferless routers. Our work can be extended in two directions.

6.2.1.1 Providing QoS Assurance in Low-cost NoCs

Firstly, heterogeneous NoCs can be employed to provide low-cost networks with QoS assurance. Because techniques targeting NoC QoS all require buffers to order packets at each router, pure bufferless NoCs cannot provide QoS guarantee for performance. Since there is no buffer in the routers to order the packets, the bufferless NoC proposed in [107] employed hot potato routing protocols. If there is contention in the output ports, the packets are deflected to another output port that has no packet scheduled for it. To avoid deadlock, they use age based mechanism to give older packets higher priority in outport arbitrations. However, such a mechanism cannot provide user dataflow based prioritization. We can extend our heterogeneous NoC design that employ bufferless and buffered routers to provide QoS. Bufferless routers are used to reduce cost and buffered routers can be equipped with QoS queues. Figure 6.1 depicts the high level view of such a heterogeneous network. In such a network, the bufferless routers are designed similar to the bufferless routers proposed in [107]. The buffered routes are equipped with priority buffer queues. Incoming packets are put into different buffer queues based on their QoS related priority. At the outport arbitration stage, packets from the buffer queue with the highest priority are granted the outports first. The number of buffers in the priority queues is closely related with the level of QoS that can be provided by such a network. In addition router placement plays an important role in such an NoC architecture, similar to what we have shown in Chapter 4.

Such a network does not provide QoS guarantee at each router like other complicated designs that have been proposed to provide NoC QoS. As a result, it may not provide tight latency or bandwidth bounds like other schemes. However, it provides QoS to applications at a certain level and the level of QoS provided can be adjusted with the number of buffered routers and their placements. Such a design will achieve better tradeoffs between the service provided by the NoC and
6.2.1.2 Improving Reliability in Low-cost NoCs

Other than providing QoS, we can also exploit the NoC heterogeneity to enhance the reliability of bufferless NoCs. As discussed in Chapter 3, with the help of ECC codes, packet errors can be detected in the middle of path to the destination and get retransmitted from a correct copy. However, because bufferless NoCs have no buffers at each router, retransmission has to start from the source of the packet. By adding buffers to some of the routers, we can save a correct copy of the packet in a router closer to the destination. Thus we can not only reduce the packet retransmission latency but also decrease the power consumption. Figure 6.2 illustrates the packet retransmission scheme in bufferless NoC and the heterogeneous NoC we propose. In Figure 6.2(a), even the error can be detected when a packet almost reaches the destination, the data retransmission has to start from the source node, since there is no copy saved in routers on the path. In the heterogeneous design shown in Figure 6.2(b), some routers are equipped with retransmission buffers that keep a copy of the packet. So the retransmission can start from such midway
Figure 6.2: Packet retransmission with (a) bufferless NoC or (b) heterogeneous NoC.

6.2.2 Exploiting program parallelism to increase power-efficiency for multiprogrammed applications

The technique that dynamically configures the number of cores and cache sizes described in Chapter 5 can be combined with other power management techniques. For example, DVFS has been a mature technique in power management and widely applied to modern processors. Our dynamic resource configuration scheme of TaPER can be combined with DVFS techniques. For example, instead of gating off some cores or caches, we can further explore the design space by applying DVFS to those components.

In addition, with TaPER, we only evaluated the technique with single application running multiple parallel threads. It would be common for multiple multi-threaded applications to run on a single chip especially when more cores are avail-
able. Since applications have different scalability with thread counts and cache sizes, it would be more challenging about how to efficiently allocate resources to the threads of each program. For example, we need to first partition the cores and caches among several applications based on power constraints. Then for each application, we need to figure out a configuration that will result in best performance for this application. The partition of resources needs to be dynamic in order to capture the changing behavior of applications at run time. The benefit of such schemes would be maximized performance gain with minimal energy overhead.

Figure 6.3: Allocate core/cache power to multiple parallel applications based on the programs’ scalability.

Figure 6.3 shows a scenario where multiple parallel applications run on a many-core multiprocessor. There are three applications, and each is running with multiple threads on some cores and are allocated with some portion of the total cache. During the execution, the available total power changes and causes a power emergency. We can change the number of cores or size of caches allocated to each application based on their scalability and still achieve optimal performance under this new power constraint. For example, our scheme detects that application 1 scales with number of cores, but not with the size of cache. We then apply a core-favoring power reallocation to application 1. We find the performance of application 2 does not scale with cache sizes, so we can remove some of its cache. In contrast, application 3 scales with cache sizes but not with the number of cores. So we remove some cores allocated to application 3 but give it more cache. By reallocating power based on each application’s scalability, we can achieve high power efficiency without compromising the performance for the parallel applications running together.


Hui Zhao is from the city of Harbin in northeastern China. She obtained her bachelor’s degree in Computer Science and Engineering from Harbin Institute of Technology, Harbin. She received her master’s degree in Electrical and Computer Engineering from the Northeastern University in Boston. She then worked as a semiconductor engineer in Teradyne Inc, Boston. She joined the doctoral program in the Department of Computer Science and Engineering at the Pennsylvania State University in 2008. Her research interests include several aspects of computer architecture designs, such as NoC design, reliability, and performance and energy optimizations.