Acknowledgement

I am indebted to my adviser John Wawrzynek, and members of my committee Jan Rabaey, Paul Wright and Borivoje Nikolic who have inspired and motivated the theme of this research effort.

The work was supported by the GSRC Focus Center, one of five research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program, and by Berkeley Wireless Research Center (BWRC).

A System-level Approach to Fault and Variation Resilience in Multi-core Die

by

Yury Markovskiy

B.S. (University of California, Berkeley) 2000
M.S. (University of California, Berkeley) 2004

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in Charge:

Professor John Wawrzynek, Chair

Professor Jan M. Rabaey

Professor Paul K. Wright

Fall 2009

A System-level Approach to Fault and Variation Resilience in Multi-core Die

Abstract

A System-level Approach to Fault and Variation Resilience in Multi-core Die

Yury Markovskiy
Doctor of Philosophy in Computer Science
University of California, Berkeley
Professor John Wawrzynek, Chair

With shrinking transistors and growth in parametric variability, statically managing die yield is no longer possible. Design for Manufacturing (DFM) techniques use increasingly bigger guard-bands that waste area, power, and performance, impeding Moore's Law of semiconductor device scaling. Process Voltage Temperature (PVT) variations can turn a nominally homogeneous many-core die into a set of cores with heterogeneous performance.

Network-on-Chip provides an effective and scalable way to integrate hundreds of heterogeneous cores without forcing each to give up its own PVT-induced operating point for the chip-wide common worst case. As with asynchronous logic, a NoC of regular, redundant, many-CLK/V_{DD} cores can deliver the average rather than the worst case system performance with greater power efficiency and fault tolerance than its globally synchronous monolithic counterparts [41, 92]. This work shows that the Voltage-Frequency Island (VFI) architectures are also the key to tolerating and compensating for PVT variations.

The VFI advantages cannot be realized without run-time task-to-core mapping and adaptive network routing that optimally match application resource

requirements with heterogeneous cores and communication fabric. These systematic techniques are more effective at mitigating a variety of faults and variations than layout and circuit DFM. Most importantly, the gains from these techniques can be translated into die yield improvements and smaller DFM guard-bands.

This work investigates core sparing and network routing. The developed models demonstrate that core sparing reduces the die cost asymptotically from $O(A^3)$ to $O(A^{\frac{1}{2}})$, and it is more cost efficient than larger design guard-bands of layout and circuit redundancy. The analysis outcome favors a greater number of smaller unreliable cores as opposed to a fewer larger reliable cores given a fixed die area. This points to the limitations and ultimately the futility of DFM techniques in the future semiconductor process generations.

Adaptive network routing enables core sparing. More critically, it simultaneously combats the two sources of network load imbalance: on-die performance heterogeneity from PVT variations and application communication topology. With stochastic PVT variations, the developed Minimal Adaptive Total Congestion (MATC) router increases the expected network saturation bandwidth by 7–23% and reduces its variance by 2–10x as compared to the Dimension Order router. With systematic PVT variations, the improvements are 5–35%. These gains of the adaptive router can compensate for degradation due to performance variations and can thus be used to reduce design guard-bands.

By treating cores as units of fault and variation tolerance, these systematic techniques provide a simple and consistent way to deal with static and dynamic performance variations and faults. These techniques are more effective than isolated DFM solutions. Rather than fighting and minimizing the on-die parametric variations, our approach takes advantage of the platform heterogeneity and manages its net system performance impact.

Professor John Wawrzynek
Dissertation Committee Chair

Contents

List of Figures

List of Tables

Acknowledgements

This work is the result of generosity and support of many colleagues and friends who have made my experience in U.C. Berkeley a memorable one.

I would like to acknowledge the old SCORE gang, my friends Randy Huang, Eylon Caspi, Joe Yeh, Nick Weaver who have influenced me immeasurably, showed me what research is all about. I would like to thank André DeHon, whose drive for the scientific truth always served as my inspiration. Enormous debt of gratitude goes to my friends Yatish Patel, Mark Whitney, Sonesh Surana, Nemanja Isailovic for wonderful conversations, exchange of ideas, and helping hands. I am very grateful to my friend Michelle Bird who has helped enormously to communicate my ideas through this manuscript.

I am indebted to my adviser John Wawrzynek for the continuing support, encouragement and freedom to explore many interesting ideas throughout my graduate work. My dissertation committee Jan Rabaey, Paul Wright and Borivoje Nikolic have inspired and motivated the theme of this research effort.

MPI application traces, used in this work, were kindly provided by Shoaib Kamil and John Shalf from Future Technologies Group at the National Energy Research Scientific Computing Center (NERSC) at Lawrence Berkeley National Laboratory. The data traces were collected as a part of DOE/LBNL funded research. My work was supported by the GSRC Focus Center, one of five research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program, and by Berkeley Wireless Research Center (BWRC).

And most importantly, none of this work would even be possible without love, support and encouragement of my parents Mark and Irina, my brother Igor, and my wonderful wife Carmen, to whom this is dedicated.

Curriculum Vitæ

Yury Markovskiy

Education

2004 Masters of Science in Computer Science, University of California, Berkeley

2000 Bachelor of Science in Computer Science and Electrical Engineering, University of California, Berkeley

Experience

- Graduate Student Instructor, UC Berkeley (Spring and Fall 2002)
- Programmer Analyst II, UC Berkeley (2000 – 2001)
- Software Development Engineer, ProxiNet, Inc., Emeryville, CA (1999)
- Software Development Engineer, Canary Technology, Fremont, CA (1997 – 1998)

Selected Publications

- Yury Markovsky, Yatish Patel, John Wawrzynek. "Using Adaptive Routing to Compensate for Performance Heterogeneity" in International Symposium on Networks-on-Chip (NOC), San Diego CA, May 2009
- Yury Markovsky, John Wawrzynek. "On the opportunity to improve system yield with multi-core architectures" in IEEE International Workshop on Design for Manufacturability and Yield (DFM&Y, Santa Clara, CA, October 2007).

- Jana van Greunen, Yury Markovsky, Christopher R. Baker, Jan Rabaey, John Wawrzynek, Adam Wolisz. “A Platform for Smart Home Environments: The Case for Infrastructure” in Proc. of the 2nd International Conf. on Intelligent Environments (IE06, Athens, Greece, July 2006).

- Christopher R. Baker, Yury Markovsky, Jana van Greunen, Jan Rabaey, John Wawrzynek, Adam Wolisz. “ZUMA: A Platform for Smart-Home Environments” in Proc. of the 2nd International Conf. on Intelligent Environments (IE06, Athens, Greece, July 2006).

- André DeHon, Yury Markovsky, Eylon Caspi, Michael Chu, Randy Huang, Stylianos Perissakis, Laura Pozzi, Joseph Yeh, John Wawrzynek. “Stream Computations Organized for Reconfigurable Execution” in the Journal of Microprocessors and Microsystems 30 (2006) 334-354.

- Nicholas Weaver, Yury Markovskiy, Yatish Patel, and John Wawrzynek. “Post Placement *C*-slow Retiming for the Xilinx Virtex FPGA” in Proc. of the Eleventh ACM International Symposium on Field-Programmable Gate Arrays (FPGA 2003, Monterey, CA, Feb. 2003).

- Yury Markovskiy, Eylon Caspi, Randy Huang, Joseph Yeh, Michael Chu, André DeHon and John Wawrzynek. “Analysis of Quasi-Static Scheduling Techniques in a Virtualized Reconfigurable Machine” in Proc. of the Tenth ACM International Symposium on Field- Programmable Gate Arrays (FPGA 2002, Monterey, CA, Feb. 2002).

- Eylon Caspi, Randy Huang, Yury Markovskiy, Joseph Yeh, André DeHon and John Wawrzynek. "A Streaming Multi-Threaded Model" in Third Workshop on Media and Stream Processors (MSP-3, Austin, TX, Dec 2001).

Chapter 1

Introduction

With the field of semiconductor manufacturing entering its sixth decade, many industry observers have anticipated exponential growth in chip performance to follow the gains in transistor and interconnect miniaturization. The reality is, however, that these gains have not yet been realized in the submicron process nodes. To state the problem in simple terms, additional efforts in miniaturization have resulted in disproportionately reduced die yield and increased manufacturing costs. We have reached diminished rates of return. Both the reduced yield and increased cost result from the fact that the smaller the transistor size, the more susceptible it is to an array of problems. Those problems include manufacturing faults, performance degradation and parametric variations.

The problems arising from semiconductor process scaling can be divided into two related categories: (1) limitations of on-chip power distribution and dissipation, and (2) Process-Voltage-Temperature induced variations.

The power consumption problems have been addressed in circuits through a variety of power management techniques that include Dynamic Voltage and Frequency scaling, power shutoff to inactive circuit modules, and transistors with different threshold voltages on a die. Reaching the power consumption ceiling has forced a more fundamental change—a movement from high clock frequency single core processors to lower frequency multi-core systems that exploit parallelism to scale overall performance.

Addressing PVT variations and faults in deep sub-micron processes has proven to be a significant challenge. Currently, attempts to minimize their impact amounts to large design margins that effectively push the technology back to larger geometries. As critical transistor sizes approach only tens of atoms and the process engineers can no longer control their manufacturing parameters accurately, these efforts hit the diminishing rate of returns. A solution is needed that would minimize over-engineering and extend performance scaling further. In other words, the solution that would tolerate and even take advantage of PVT variations in smaller geometry transistors.

1.1 Moore's Law: a blessing and a curse.

1.1.1 End to Moore's Law scaling?

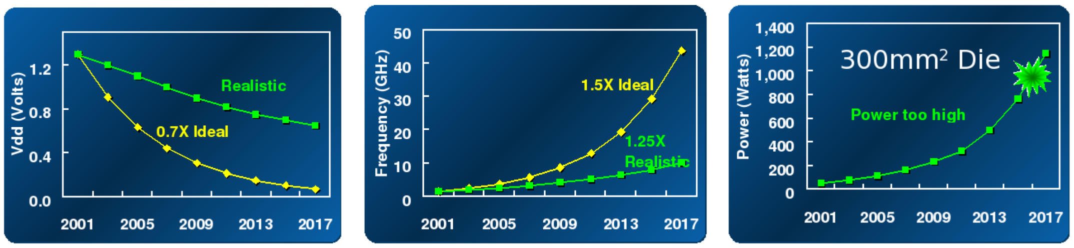
In 1965, Gordon Moore observed the number of transistors that can be placed cost-effectively on a die doubles approximately every two years [100]. Traditionally, this decrease in on-chip transistor size was accompanied by a corresponding reduction in circuit cycle time and power consumption. Originally, this trend was expected to continue without a limit. In what may have become a self-fulfilling prophesy, transistor sizes decreased over the years, and in fact, the number of transistors on a modern die exceeds two billion [129]. Unlike transistor count, however, scaling of many other critical transistor parameters has stalled as shown on Figures 1.1 and 1.2 [19, 55]. As modern sub-32nm processes are pushing against the physical manufacturing limits and properties of the silicon substrate, transistor gate oxide thickness T_{ox}, supply V_{dd} and threshold V_{th} voltages are most affected. Critical device sizes are approaching tens of atoms and can no longer be reproduced *accurately* on a die. The result is significant parametric variations in saturation and static currents, and capacitances that directly impact circuit performance. Small devices exacerbate the scaling problem further as they are more susceptible to faults and performance degradation due to aging (*e.g.* NBTI [117]).

In other words, although many small transistors fit on a die, each of those transistors is now less reliable, less performant, and more power-hungry.

In the prevalent synchronous circuit design style, device parametric variations reduce performance predictability. This translates into uncertainty in the circuit delay at the design time, which unduly inflates the clock cycle time. Growing power consumption and limits on heat dissipation present an additional constraints for clock frequency scaling.

Clock frequency has been the main VLSI performance scaling vehicle for more than 30 years; but now, to overcome power and parametric variation challenges, the industry is turning toward spatial parallelism with multi-core architectures such as Sun Niagara [102] and Intel Merom [116]. Even if an ideal compiler could extract all application parallelism, and a multi-core architecture could exploit and realize it, then the uncertainty of variations would still impose limits on power efficiency and system performance. The result would be a multi-billion transistor die that cannot deliver on its full potential.

As semiconductor devices shrink further, the increase in variations and faults makes it increasingly difficult to produce a reliable working system that meets the specification target under a range of dynamic operating conditions including different cross chip activity levels, temperature gradients and supply voltage distributions. For example, [4] demonstrated that a 35nm bulk transistor modeled



(a) Supply Voltage (b) Clock Frequency (c) Power Consumption

Figure 1.1: Ideal Moore's Law *vs* the actual and projected scaling [19].

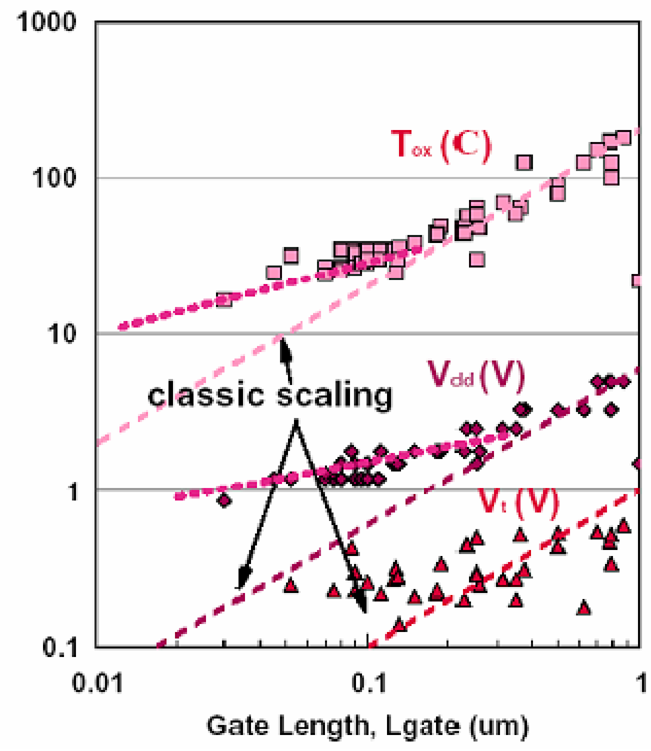


Figure 1.2: Ideal Moore's Law *vs* the actual T_{ox}, V_{dd}, and V_t scaling [55].

after a published Toshiba device [69], has more than 200mV of threshold voltage variation, and 100x leak current variation solely due to random dopant fluctuations at a single operating point. If you consider this transistor over the complete range of supply voltages and temperatures, its saturation and static current would vary widely and render some devices completely unusable due to high leakage or high threshold voltage. These large device parametric variations adversely affect transistor performance in a typical operating regime. As Figure 1.3 shows, supply voltage scaling is severely limited by transistor threshold voltage variations that are widening primarily due to random dopant fluctuations and line edge roughness (LER). Both effects are stochastic, difficult to control, and their impact worsens in smaller devices [39]. The transistor variations translate into asymmetric gate delay variations ranging from low delay, high leakage gates to high delay, low leakage gates. FMAX circuit performance model [22] and ITRS 2005 [1] predictions point to the maximum operating frequency varying by more than 70% from the mean in sub-32nm processes. Figure 1.4 illustrates the degrading effect that transistor size scaling has on the expected maximum circuit frequency.

1.1.2 Minimizing Variations

Faced with a collection of devices with very diverse performance, the semiconductor industry challenges designers to produce reliable systems out of very

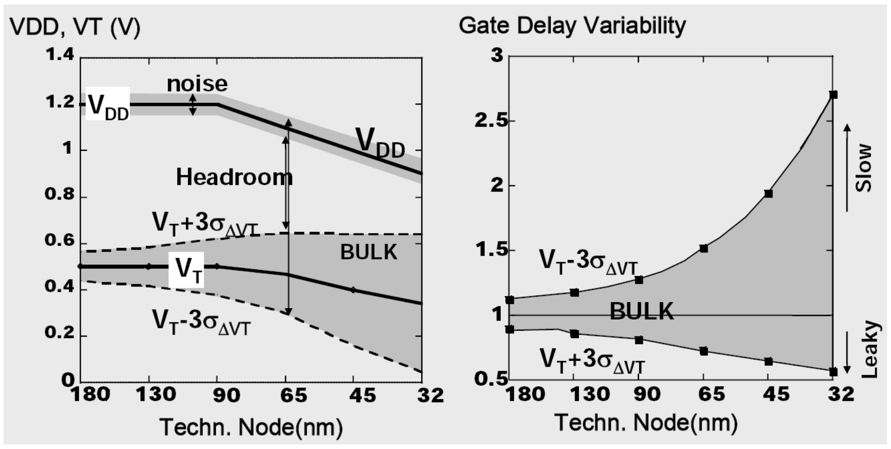


Figure 1.3: Variations and scaling effect on transistor and gate performance [39].

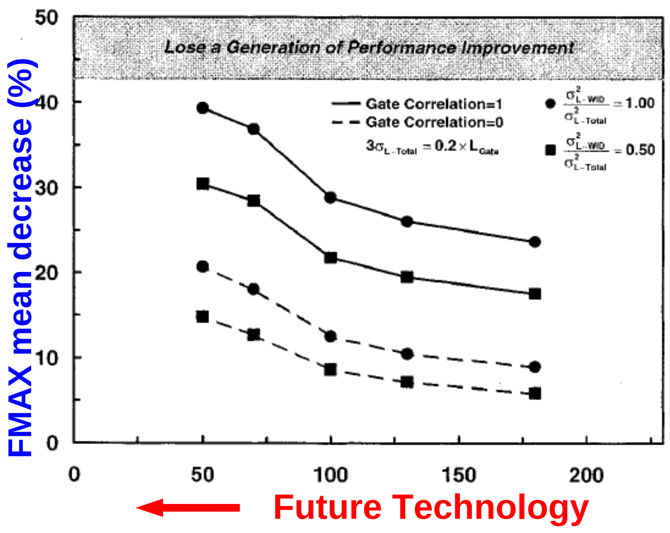


Figure 1.4: Decreasing device sizes result in parametric variations that degrade maximum circuit operating frequency FMAX [22].

unreliable and unpredictable components. Alternative post-Silicon technologies, such as dense carbon nanotubes [40], do not solve but exacerbate this problem further as they require stochastic assembly. Most likely all future systems must be built out of components whose performance and fault model can best be described as stochastic.

In an attempt to mitigate faults and variations, designers rely on Design for Manufacturing (DFM) techniques in VLSI layout, circuit "rules of thumb," and other established safe design practices. Since the sources of process variations cannot be accurately controlled due to the very small device sizes, DFM rules take a *safe* approach and force design margins that grow as devices shrink, *i.e.* larger device sizes, spacings, metal fills, and other layout modifications that the original circuit designer may not be aware of.

As yield is the main concern for fabrication engineers, they impose increasingly restrictive and complex layout design rules. In spite of their complexity, these rules do not effectively convey the nature of underlying physical processes and cannot be applied selectively by the circuit designers. They increase device feature sizes and effectively push transistors back to a previous semiconductor process generation, surrendering the expected improvements from scaling.

In addition to DFM layout rules, circuit design techniques, such as adaptive body biasing [133] and adaptive supply voltage [29], can be applied to most circuits because they simply realign device parameters from different dice to minimize variations. Other techniques, such as error correction, can be used efficiently only in some domains, *e.g.* memory arrays and communication circuits [109, 137].

As these layout and circuit techniques are applied to a specific design, their individual benefits on the parametric yield are hard to quantify. Their benefits may overlap and thus result in wasted power and performance. Fundamentally, there is a disconnect between the problem and the available solutions. The variations arise at the molecular level. DFM and circuit techniques, however, attempt to *minimize* the problem applying large margins and guardbands at the layout and circuit levels. It is similar to using a sledge hammer where a scalpel is needed, but a molecular scalpel does not exist. The result is that they reduce system performance to the worst-case operating point that accomodates all components including the outliers.

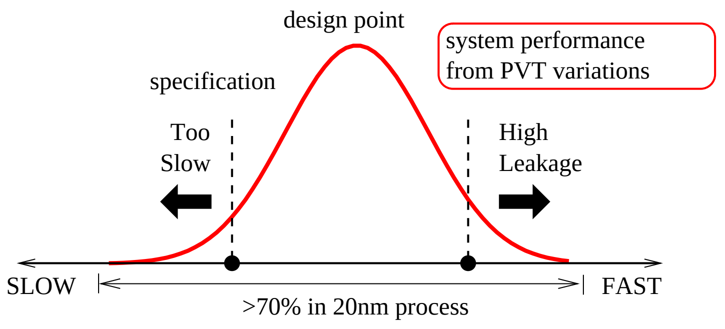
Due to trade secrets, very few published efforts have quantified the area, power, and performance losses due to DFM "over-engineering." Anecdotal evidence, however, suggests that DFM has a significant impact. One such effort has demonstrated 12% area reduction in 90nm and 65nm standard cell designs when the performance guard-bands are reduced (narrowed) by 40% [74]. The authors developed a die yield model containing area dependent and area agnostic components to illustrate the trade-off between guard-bands and the die area, which naturally corresponds to power consumption and performance.

If we had techniques that instead of narrowing the guardbands could dynamically embrace the heterogeneity and manage system components with wider performance variations, we could further reduce the area beyond these 12% and increase the die yield.

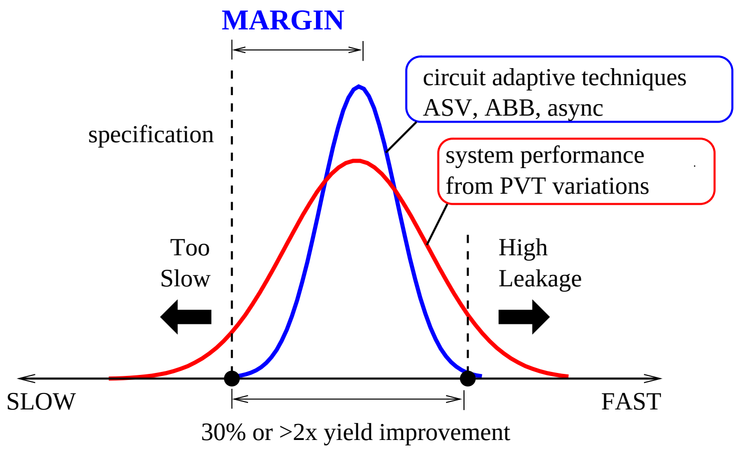
1.2 Our Novel Approach

This work's focus is the development and analysis of *system-level techniques* for dealing with faults and variations. Instead of attempting to narrow the spread of transistor and interconnect performance variations, is there a solution that tolerates a wider range of fluctuations? Can we compensate for rather than minimize parametric variations?

Figures 1.5 and 1.6 illustrate our common understanding of *system* performance variations as resilience techniques are applied. After DFM driven manufacturing, the dice and their components have a wide performance variance (Figure 1.5(a)). Some components may be considered faulty if they fall outside of the specified operating bounds. Circuit adaptive techniques narrow the variance to

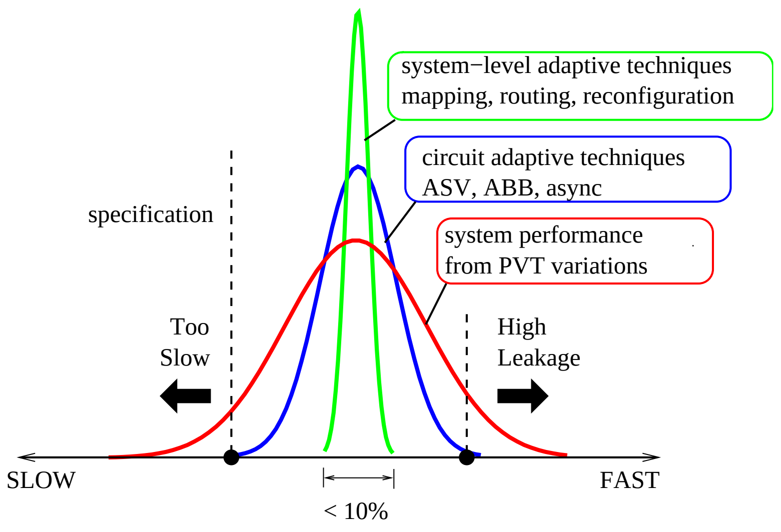


(a) After Manufacturing

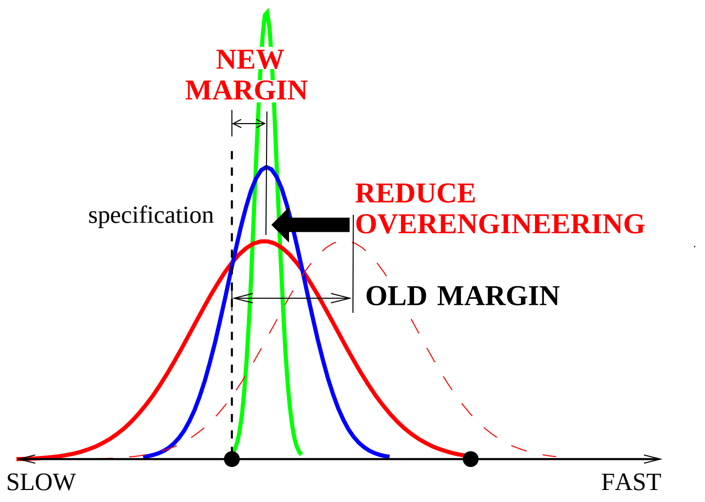


(b) Compensate with circuit adaptivity

Figure 1.5: PVT variations impact performance distribution on the current state of the art VLSI.



(a) Run-time system techniques



(b) Smaller design margins

Figure 1.6: The impact of PVT variations is reduced with multi-level techniques and translated in smaller design guardbands.

meet the targeted parametric yield and bring many outliers within the specification bounds (Figure 1.5(b)).

This work demonstrates that with run-time application mapping, network routing and reconfiguration, architects can further reduce the performance variance in multi-core systems and benefit from the proportionally smaller design margins (Figure 1.6(a)). Theoretically, with these high level techniques, the observed core-to-core variance can be reduced to 0% if the unit of resiliency management is small [118]. In practice, the improvements are limited by reconfiguration granularity and mapping quality. This work investigates resilience management at the core level, and thus small simple cores with independent clock and voltage domains that expose on-chip variations can reap the most benefit.

Traditionally, reliability has been treated as a binary problem: a component is either functional or not due to hard faults. However, every die and its components actually fall within a performance continuum: faulty, slow, nominal, fast, or leaky — where the last two may exceed the acceptable static leakage bound and be considered faulty. With respect to the execution model with core state check pointing, a faulty core is no different from a very slow processor that makes little forward progress, or a very fast but leaky processor. The advantage of our high-level approach is to treat faults as a special case of Process-Voltage-Temperature (PVT) induced performance variations, rather than as separate effect.

This systematic, consistent approach to resiliency simplifies high-level mapping and routing algorithm implementation, eliminates special corner cases, and allows designers to focus on converting the performance gains from quality mapping and routing into smaller design margins or guard-bands. It is not possible in practice to eliminate all DFM rules that result in over-engineering, lost performance, area, and power. However, as we show our high level techniques can reduce the "6σ" guard-bands, which can simplify and eliminate a number of DFM rules, improve system power efficiency and cost as illustrated on Figure 1.6(b). The resulting system can operate closer to its nominal performance point instead of the worst case point dictated by the underlying semiconductor process.

1.3 Multi-core and Reliability

Clock frequency scaling is no longer a practical tool for processor performance growth. The move to multi-core architectures and the desire to tolerate high memory latency are pushing the industry toward simpler and smaller in-order cores that time-multiplex multiple threads and operate on relatively low clock frequencies, such as Sun Niagara [102]. This means that cores are becoming cheaper. Cores are growing in numbers, and most importantly, are becoming a viable unit of reliability, power and performance management.

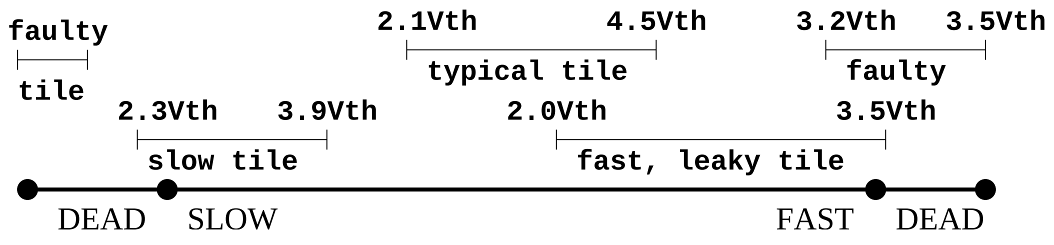


Figure 1.7: An illustration of core performance ranges controlled by DVS/DFS.

PVT variations turn a nominally homogeneous processor array into a collection of cores with heterogeneous performance [68], where the optimal power / performance point of each core is determined by its process variations and aging effects — discussed in detail in Chapter 2. A designer can choose to reduce clock frequency of some cores to match it to the slowest one on chip to create uniform on-die performance, wasting considerable resources to achieve the required system performance. This approach forces a 3σ margin between the nominal design point and the target specification. The alternative is to expose the PVT variations to the run-time system by allowing each core to run at its own power/performance operating point dictated by the process corner. In practice, the voltage and temperature variations combined with the hardware controls such as Dynamic Voltage and Frequency Scaling (DVS/DFS), turn an operating point into a *range* as shown on Figure 1.7. The run-time system can use the hardware controls to select the core's operating point from that range to accommodate core-specific computation and communication demands in a power efficient manner.

Even without PVT variations, a nominally homogeneous multi-core system is a system where every core has different performance. First, consider that a multi-core die is a highly complex system. Its behavior will only become more complex as it exceeds 100 cores and moves toward a more scalable memory and communication fabric organization, such as distributed memory blocks and stylized networks. As a result of contention for resources, cores suffer from unpredictable cache behavior, limited memory bandwidth and widely varying communication latency. Second, every application task may have very different computation and communication resource requirements and correspondingly affect every core. The problem is difficult to remedy, as parallelizing compilers are still relatively immature and cannot deliver the fine-grained parallelism needed for near-optimal load balance. Balancing and scheduling of communication traffic, memory and I/O accesses, in addition to the traditional processor scheduling, makes the compiler parallelization problem even more intractable.

An attempt at a performance homogeneous multi-core architecture would result in an inherently inefficient and slow implementation. Such an architecture would contain a uniform access memory structure, no caches, and require *a priori* optimally scheduled and balanced loads, which is not suitable for a system with large number of cores. To efficiently utilize massively parallel hardware, it must be heterogeneous to deliver on the architecture performance potential.

System Layers	**Permanent faults, variations**	**Dynamic variations**
Application		Error tolerance
Mapping	Run-time Adaptive Routing and Mapping	
Routing		
Topology	Redundancy and Scalability	
Link	Fault-tolerant control flow	Adaptive links [137]
Circuit	Self-checking circuits [146, 62]	ABB, DVS [59, 132]

Table 1.1: Every layer of a multi-core NoC system plays a role in dealing with permanent and dynamic variations

Presently, the performance of a realistic scalable system is typically best described with stochastic predictive models [70], which accommodate a number of factors given a probability distribution of critical events in an application. At the same time, PVT variations result to a significant degree from the inability to control stochastic semiconductor processes. The variations create a complex core performance profile that itself can most effectively be described with stochastic models. Therefore, why not combine both stochastic models together and simply treat *any* multi-core system as a distributed stochastic system?

A comprehensive resilience solution has to come from a complete cross-layer approach outlined in Table 1.1. Every layer has the potential to deliver an efficient solution to its part of the problem. For example, power-adaptive error resilient communication links may be most efficiently implemented at the circuit link layer. They require a retransmission scheme that should ideally be a part of an application or router implementation and not built into the circuit implementation

Design Time	**Run time**
• circuits to *detect* fault/variations and *adapt* • layout to minimize impact of variations • provide run-time knobs	• Map and route around faults and "slow" components • Optimize latency and B/W • Degrade power/performance gracefully

Table 1.2: Variation compensation techniques must be judiciously applied at design and run time.

to avoid wasting buffer resources for a relatively infrequent event. Applications or routers are better suited to handle retransmission because they are aware of system-wide communication QoS requirements. The challenge is to understand how to minimize the overlap between techniques from different system layers and reduce "over-engineering." The overall strategy is a combination of design time architecture and circuit decisions with the run-time adaptive techniques, outlined on Table 1.2.

Our answer to fault and performance variation resilience is run-time resource management: adaptive task-to-core mapping and routing, which ultimately is a solution to any performance heterogeneous configurable platform. If an architecture also provides hardware mechanisms for core-level fault detection and component reconfiguration, the multi-core system can also operate closer to its optimal power / performance regime. The run-time environment, *e.g.* an operating system or a hyper-visor, can manage resources using these hardware mechanisms.

It can reduce the impact of PVT variations better than disparate ad hoc point solutions because it is simultaneously aware of application resource *requirements* together with the hardware *utilization* and its on-chip performance profile. Over-performing cores and interconnect can compensate for under-performing ones to result in average rather than worst-case system performance.

The run-time management can be driven by analytical models or system introspection or a combination of the two. The analytical models may give insight into system behavior but lack practical applicability due to their limited accuracy, as is the case with PVT variation modeling. With process scaling, the contribution of hard-to-quantify and hard-to-model effects is likely to grow in comparison to systematic, modelable ones. Additionally, the criticality of different PVT variation sources changes with applications, chip architectures, and semiconductor process maturity. Thus, predicting and compensating for different variations with a model is inefficient and intractable.

This work focuses on introspection. Introspection is the dynamic process in which the run-time system observes behavior and performance of individual cores, network routers, and the system as a whole. It then attempts to modify task mapping and routing to improve overall system performance. The introspective approach has scalability and flexibility advantages over modeling by encompassing all inter-dependent factors into relatively simple observable metrics. With performance counters and control knobs, the run-time layer can form a closed-loop system with the architecture and thus converge toward the optimal operating point for an implementation and its particular process corner.

1.4 Research Contributions

This work identifies implementation requirements for a multi-core architecture with independent processor core clock and voltage domains. This architecture enables core and NoC router performance to abstract PVT variations and to guide introspective run-time adaptive mapping and routing (Chapter 3). The adaptive techniques thus simultaneously compensate for load imbalance resulting from static process and dynamic voltage and temperature induced performance variations as well as the application specific load imbalance. Based on this architecture, the following two techniques are investigated: core sparing and adaptive network routing.

Core sparing is a strategy of implementing extra cores that are reserved to replace cores that are defective due to manufacturing and/or dynamic faults (Chapter 4). The work develops an analytical model to demonstrate that this technique asymptotically reduces the die cost from the traditional "no spares" $O(A^3)$ down to $O(A^{\frac{1}{2}})$, where A is die area. Additionally, the developed die cost models for DFM layout margins and circuit modular redundancy compare these established techniques with core sparing. These analytical models open a discussion about the trade-offs between over-engineering of individual cores, *i.e.* hardening and protecting circuits within a core *vs* relying solely on core redundancy to achieve system fault and variation resiliency. Our analysis shows that a die with many small unreliable cores is more cost effective than a die with fewer larger hardened reliable cores. To use core sparing dynamically, the core must detect faults and report them to the run-time system that can relocate tasks with a check-point and restart mechanism.

Adaptive network routing is the key to realizing the potential of a globally asynchronous tiled multi-core architecture to deliver average rather than worst case network performance (Chapters 6 and 7). The architecture exposes PVT induced performance variations as network router congestion, and allows an adaptive routing algorithm to compensate for the variation by balancing communication load across the network. A congestion-aware router simultaneously compensates for static manufacturing faults and process variation, dynamic voltage and temperature variations, aging, and application load imbalance. This simple and yet effective approach is necessary in an environment where the system component performance can best be described as stochastic or unpredictable. Adaptivity to the environment and operating conditions is essential to dealing with uncertainty in the future semiconductor processes, where detailed characterization and modeling is intractable.

As a result of this research effort, we have developed a flexible, discrete event Network-on-Chip simulation infrastructure targeted at investigating the impact of PVT-induced performance variations. The configurable simulator accommodates a range of routing algorithms and permits the simulation of communication patterns from *real* applications (Chapter 5).

1.5 Related Works

The subject of this research spans a great number of topics including low level PVT variations and their sources, task mapping, network routing, and design space exploration frameworks. The following discusses many relevant efforts and their relationship to this investigation. Although several efforts discuss their proposed architectures and systems as tolerant of PVT variability, faults and heterogeneity, none has addressed or compensated for these on the system level, which is the main contribution of this work.

Variability Modeling As technology moved to sub-micron dimensions, the interest in variability modeling has dramatically increased. The published research ranges from modeling the magnitude and range of manufacturing effects such as random dopant fluctuations (RDF) [89], line edge roughness (LER) [104] that directly affect threshold voltage and saturation current in transistors, to modeling gate parameters affected by the process variations: static leakage current [60] and delay [91]. The models identify that logic gate performance is most sensitive to RDF and LER, but also is heavily dependent on the supply and threshold voltage ratio [28].

The FMAX model [22] offers a concise and intuitive way to analyze the expected critical path delay and its variance for a general circuit comprising a large number of logic gates and near critical paths. The FMAX model led to [68] that identified that the impact of within die and die-to-die variations on multi-core die is determined by the *systematic* on die variations, while stochastic effects are contained through longer critical path delay averaging within a core. The merits of the Globally Asynchronous architecture described in Chapter 3 are based on these conclusions, which also shaped our thinking about variation modeling within our simulation framework.

Adaptive circuits Although the thrust of this work is on core-level adaptivity, low level adaptive circuits can have a dramatic impact on power savings when used in certain appropriate domains, such as communication. [137] discusses a low power adaptive transmission scheme for NoC that trades off power consumption for error rate. This scheme requires a voltage controller and retransmission logic to recover from sudden changes in error rates. The supply voltage controller attempts to keep retransmissions very infrequent. This type of scheme must be tightly integrated with the NoC router logic for maximum efficiency to accommodate real time and QoS constraints in the minimal power envelope. For example, consider the case where a packet should be dropped instead of retransmitted to meet the constraints. This decision ideally must be made by the routing logic or the application.

This work also relies on different adaptation techniques such as Adaptive Body Biasing [133], voltage and frequency scaling [52]. Unlike adaptive communication schemes, these techniques are orthogonal to the functionality of a core or a router, but they dramatically impact its performance. Body biasing can realign the core performance to reduce the variance, and voltage and frequency scaling can navigate the performance operating range resulting from process variation impacts on a particular cores.

Redundancy and **Checkers** The only mechanism to combat faults and improve PVT variation resilience is redundancy. The examples of spatial redundancy include Triple Modular Redundancy (TMR) [131], row and column spares in DRAM [130], core spares in Sun Niagara [102] and Cell [77], and many others.

A simple example of temporal redundancy is Razor latch [53] that registers the value from a combinational path twice: first, on the clock edge, and second, on the delayed clock edge. By comparing the latched values, the Razor latch can detect small changes in critical path delay as setup time violations and alert the system of an error. This technique saves 20–40% of power in a manufactured Alpha processor, that implemented dynamic voltage scaling and micro-architecture error state recovery controlled by the error-rate reported by Razor latches. Similar to our work, Razor is an introspective technique that smoothly adapts to changes in voltage and temperature, as well as aging and application instruction stream, which permits the processor to run closer to the average rather than the worst-case power consumption regime.

GALS Architectures Multiple clock and voltage systems, similar to the architecture described in Chapter 3 have been proposed and hailed for their potential in fault and PVT variation tolerance [92]. However, application implementation and study of actual performance of these systems has been scarce. [103] presents an application-specific partitioning scheme of the multi-core architecture into voltage-frequency islands (VFI) comprising core clusters to optimize energy consumption. The authors experimented with one small media application comprising fewer than 10 tasks, and showed approximately 35% energy reduction. The power/performance benefits of VFIs will be more significant in systems with greater number of processor cores. This work investigates network performance for larger 64 and 256 core systems, and instead of clustering cores into VFIs, allows *each* core to select its own voltage / frequency operating regime.

Task Mapping is among the two critical run-time processes necessary to ensure computation load balance and to meet the expected performance of multi-core system. Datta *et al* showed that even small scale, four and eight core systems with Non-Uniform Memory Access (NUMA) require "location" aware task-to-core assignment for maximum performance [38]. Mapping a communicating task graph onto a system with a large number of cores requires even more considerations than NUMA optimizations, which only considers data set partition and its affinity with the computation. The mapping problem is essentially a task communication graph embedding into the network topology graph given a set of bandwidth and latency constraints. It has been tackled in different environments from the Blue Gene/L super computers with 3D network topology [143] to a heterogeneous NoC [84].

Heterogeneous architectures are gaining popularity in the multi-core research community primarily for their power-efficiency due to core specialization. Asymmetric architectures with a single common ISA shared by all cores but different complexity, heterogeneous core implementations seems to be preferred. This configuration simplifies compilation but complicates and constrains run-time task scheduling [121, 11, 30, 12]. What these efforts have not considered yet is network topology and locality aware task mapping, which is among the most critical factors for high network performance.

This work does not focus on task-to-core mapping, but offers a cursory investigation of its performance impact in Section 7.5. For the network routing experiments, we use an existing Simulated Annealing FPGA placement algorithm to compute locality preserving task-to-core mappings [16].

Network Topology determines the most basic bounds of application network performance as discussed in Chapter 6. Its on-die implementation is critically important, and thus the VLSI layout, simplicity of implementation and verification must all play a central role in topology selection. Although many sources discuss complex but asymptotically high-throughput and low latency topologies, such as Leiserson's Fat Tree or Mesh of Trees [72], this work focuses on a regular simple mesh. Mesh and its derivative Torus have been studied extensively to identify the optimal dimensionality given fixed on-chip interconnect resources [35]. As on-chip interconnect throughput scales, a delay-optimal topology implementation points to a high dimensional meshes and tori [80]. This work, however, focuses on a low radix 2D mesh because the power consumption of a high-speed on-chip interconnect can be prohibitive. A low dimensional mesh with *slower* but wider inter-router channels should be preferred. A higher radix topology maybe more appropriate for inter-chip communication, since the pin count does not scale as fast as on-die transistor and interconnect density.

A regular mesh topology can be augmented with special long range links and customized for a particular application running on a multi-core system [113]. It is not clear how to apply this technique in general. Perhaps, one can insert *regular* long range links into a 2D mesh to increase its "global" reach and improve performance for applications with non-local communication pattern as is done in FPGA configurable interconnect [124].

Although every topology can be evaluated with performance and power metrics, this work is also interested in network path diversity to enable flexible compensation for PVT variations and faults as discussed in Section 3.2.

Traffic Modeling Application traffic pattern ultimately determines whether communication resource requirements match resources delivered by the network topology and routing algorithm. A complete but costly performance analysis can only be performed on an actual implemented multi-core system, which precludes any efficient design space exploration. Modeling and synthesis of communication patterns, some of which may be unknown, is critical in order to perform large design space exploration of novel architectures. Typically, network research is performed with a set of synthetic communication patterns ranging from easy and load balanced (*e.g.* local, neighbor-to-neighbor) to global unbalanced patterns (*e.g.* tornado).

To meet the necessity for a more complex and realistic traffic patterns, Soteriou *et al* have proposed statistical traffic model that captures both temporal and spatial communication behavior [126]. The difficulty with that model spawns from the fact that the actual applications contain a combination of several communication patterns. Our work analyzes communication traces of High Performance Computing multitask kernels, and synthesizes the identical spatial traffic patterns in the developed NoC simulation framework (Chapter 5). These application traces were generated and previously analyzed by Shalf *et al* in the context of large multiprocessor supercomputers [119].

Adaptive Routing Although adaptive routing is not a novel concept, this work applies it to compensate for PVT variations across the die and deliver average rather than worst case network performance. Two published adaptive routing algorithms had a significant influence on this work: Dynamic Adaptive/Deterministic DyAD [66] and Channel Queue Routing CQR [122]. Although they apply to different topologies, both algorithm use flit buffer utilization as congestion metrics that drive distributed routing decisions. Additionally, both algorithms dynamically switch between different modes of operation: DyAD selects between deterministic and adaptive routing based on statically set congestion threshold; CQR uses a heuristic to detect network load imbalance and adjusts between a minimal and mis-routing path selections.

Fault Tolerant Routing This work focuses on adaptive routing as a vehicle to compensate for load imbalance resulting from PVT variations, but does not explicitly consider fault-tolerant routing. A congestion aware router behaves correctly with faulty nodes and routes traffic away or around them because the faulty, zero throughput nodes quickly become congested. At the same time, the routers around the faulty router also become congested. This growing congestion region would eventually result in a deadlock unless resolved externally because more buffers would contain packets waiting to be routed, but no forward progress can be made.

The way to resolve this problem is orthogonal to a routing algorithm or its ability to adapt. Detecting a fault and recovering from a fault is similar in principle to deadlock detection and recovery. It is an implementation specific decision that depends on the communication fabric guarantees. For example, if a fabric does not guarantee delivery, then simply dropping a packet on a time-out and marking unresponsive output channel as unavailable can solve the problem. This time-out and retransmit scheme must be supported by the run-time environment or the application if lossless delivery is required. A routing algorithm itself cannot make these decisions, but it simply has to adapt when an output router channel becomes unavailable.

A different approach to network fault-tolerance is stochastic routing [50], where to packets are replicated and propagated with certain probability in multiple directions, thus increasing their chances of not being dropped. This very interesting idea nonetheless requires support from the run-time system to handle arrival of multiple copies of the same packet, out of order arrivals, and a remote chance that no copy of a packet arrives to its destination at all.

Deterministic Routing using Safety Levels is an alternative to stochastic routing [139]. The algorithm consists of two components. The first one marks all nodes around a faulty node as unsafe according to special rules that logically modify network topology and guide packets away from *unsafe regions*. The latter part of the routing algorithm is adaptive. A packet is always forwarded to a safe node, if it is available, and an unsafe node otherwise.

Design Exploration Frameworks As a part of this research work, we have a developed a highly configurable Network on Chip simulation framework. The discrete event simulator easily incorporates the effects of PVT variations on NoC router components. To the best of our knowledge, no other framework that accommodates on-chip performance variations is currently available.

A cycle-stepping simulator such as ORION [136] is an effective and widely used tool to evaluate routing algorithm performance and estimate its power consumption. The value of the simulator stems from integrated tools such as Garnet [5] that provides a detailed network model of a NoC implementation or Polaris [125] that performs MPSoC/NoC design space exploration. Unfortunately, ORION models globally synchronous systems and is valuable primarily for routing algorithm research. In contrast, Beigne *et al* developed an asynchronous NOC architecture design framework [10] that can produce GALS chip implementations, but has not demonstrated power/performance advantages over the equivalent, globally synchronous architecture. Our research effort bridges the gap, providing the infrastructure to study GALS network systems with real workloads, routing algorithms, switching schemes and other architectural parameters.

1.6 Dissertation Organization

This dissertation is organized as follows.

Chapter 2 discusses the sources and modeling of on-chip performance variations and their impact on multi-core die performance. The chapter highlights both the impact and limitations of a range of existing Design-for-Manufacturing (DFM) and circuit techniques designed to mitigate PVT variations.

Chapter 3 presents the salient features of a core-level Voltage Frequency Island (VFI) architecture and globally asynchronous Network-on-Chip motivated by the on-die variation models.

Chapter 4 discusses the use of redundancy for fault tolerance at several system levels and develops analytical models for die yield and cost. The models demonstrate that core sparing is the most cost effective spatial redundancy as compared to larger layout margins and circuit module redundancy within a core.

Chapter 5 presents our Network-on-Chip simulation infrastructure used to investigate the network performance impact of PVT variations and the ability of adaptive routing algorithms to compensate.

Chapter 6 discusses network performance bounds and develops several adaptive routing algorithms and output channel selection heuristics to identify the overall best router for our set of High Performance Computing benchmarks.

Chapter 7 presents a new PVT variation network performance bound to provide a theoretical foundation for the network performance impact of PVT variations. It continues to demonstrate that a simple adaptive routing algorithm effectively compensates for the performance degradation. The chapter concludes by presenting a method to convert the performance gains of adaptive routing into reduced design margins and evaluates the impact of task-to-core mapping on network performance.

Chapter 8 presents two analytical models that enable a designer to efficiently evaluate real application performance on 2D planar network topologies and compares the analysis results to those obtained from NoC simulation.

Chapter 9 summarizes this work and highlights critical lessons and trends that dominate the results obtained.

Chapter 10 discusses some unanswered questions and interesting research directions that are based on this investigation.

Chapter 2

Performance Variations

Many researches have attempted to characterize parametric variations that result from modern semiconductor processes, and suggest appropriate actions that would reduce these variations and improve die parametric yield. These techniques are aggregated into a set of guidelines known as Design-for-Manufacturing (DFM). DFM rules grow in number and complexity with device scaling, but collectively remain inadequate to compensate for the variations, since they are applied ad hoc and solve a limited problem without any regard for other factors [79]. The impact of parametric variations continues to increase.

Attempts to characterize the performance impact of process variations at transistor and gate level resulted in detailed performance models [26, 18]. It suffices to say these models are inadequate because their authors out of necessity make simplifying assumptions on device parametric distributions. In practice, this limits applicability of these models to predicting trends and making design recommendations. They are not able to accurately estimate system performance, which is something that architects need.

The complete variability picture is significantly more complicated than useful models can describe or DFM can address. No one but the experts in the semiconductor fabrication facilities have a good understanding of the variations in the current and future processes. The ability to control stochastic processes such as etching, chemical deposition or lithography is fundamentally limited by physics. Therefore, DFM guard-bands and margins, or simply a lot of "over-engineering" remain the method employed to produce state of the art designs today.

This chapter summarizes physical sources of Process Voltage Temperature variations, their relationships and correlations, and develops a simple approach to modeling Network-on-Chip router performance from basic transistor parameters. The performance variation profile of a multi-core system is developed to demonstrate the foundation and the motivation for the tiled multi-clock architecture. An understanding of the relationship between the variations and system performance helps to evaluate requirements for higher-level, introspective methods such as sparing or routing that can minimize DFM over-engineering.

2.1 Sources and Trends

2.1.1 PVT Variation Sources

The impact of semiconductor devices parameter variations on the device performance is a reflection of the physical world, an environment where no two components can be manufactured with exactly the same precision. Since their inception, semiconductor processes have encountered problems with variations. As processes matured, every device generation node was tuned for mass manufacturing in spite of parameter variations. Unfortunately, the critical dimension (CD) of the current generation of devices are comparable in size to the molecules that are manipulated during the semiconductor process. For example, the channel of the smallest size transistor contains less than 100 implanted ions; the low level metal interconnect is so thin that a few missing copper atoms dramatically change its electrical properties. The semiconductor manufacturing processes are naturally stochastic and rely on *large* quantities of atomic particles for accuracy and repeatability. As the number of particles per device decreases, the variance on device parameters increases sharply. This is illustrated by the following relationship [22]:

$$\sigma_{V_{th}} \sim \frac{1}{\sqrt{W_{eff} L_{eff}}} \tag{2.1}$$

which states that the standard deviation of transistor threshold voltage, the key parameter that determines both the saturation current and the static current, is inversely proportional to the device size. Semiconductor process induced variations are just one source of concern on the modern die. Due to the die size and device density combined with integration of multiple functionally heterogeneous components, it is no longer possible to accurately predict temperature and voltage gradients across the chip [101]. This section discusses the complex inter-relationship between the three sources of performance variations: Process, Voltage and Temperature (PVT).

Aggregate PVT variations can be measured as delay or current on a large number of nominally identical devices and structures. This has enabled many researchers to classify and understand the correlations between various semiconductor device parameters for generations of semiconductor process nodes [107]. The parametric variations can be split by scope into wafer-to-wafer (W2W), die-to-die (D2D), and within die (WID) components [133]. Understanding of the impact of each of these is critical to find the best method to address them. The general approach to compensate for W2W and D2D parametric variations, such as spread in transistor threshold voltage V_{th}, is to employ chip-wide techniques. Examples of chip-wide techniques are post manufacturing or dynamic adjustments to body bias and/or supply voltage to "align" gate delays and power consump-

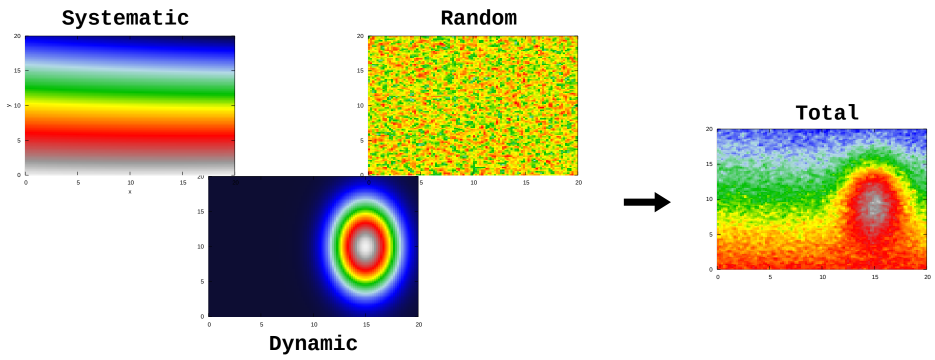


Figure 2.1: A die performance profile comprises stocastic and systematic process effects and a dynamic gradient.

tion across multiple dice. Within die variations have to be addressed differently depending on the source and nature of the variations: systematic, stochastic, and dynamic that are illustrated on Figure 2.1.

Systematic variations can be traced to predictable sources and repeatable effects that theoretically can be compensated during the system design, if they can be characterized. This characterization requires a mature semiconductor process. Initially in the process development, a number of variables are changing simultaneously, which makes it difficult to isolate and identify systematic variation sources from random and dynamic ones. The systematic sources include lens aberrations, sub-wavelength optical lithography requiring optical proximity correction, movement direction of the ion implanter and lithography steps, pattern-density [107]. All these process effects are the result of physical limitations.

Consider sub-wavelength photo-lithography that requires Optical Proximity Correction. During each mask preparation step (initial synthesis, DFM transformations and OPC), CAD tools introduce systematic errors in layout geometries that accumulate to a significant variation from the nominal device parameters. This is simply a limitation of the way modern, scalable CAD flows are implemented as a sequence of *independent* optimization algorithms. In addition to the semiconductor process, chip architecture and its on-chip layout create a number of systematic performance variations. For example, a chip layout may create temperature and voltage hotspots, essentially guaranteeing that different parts of the die will operate in different performance regimes.

Stochastic variations are inherently unpredictable and hard to control, although their magnitude can be managed as discussed in Section 2.2. The examples of random variations include line edge roughness (LER), ion dopant density fluctuations, presence of impurities, gate length and transistor oxide thickness [104]. They typically affect key transistor parameters such as threshold voltage and saturation current. These effects have the maximum impact on the smallest size transistors as illustrated by Equation 2.1. This severely limits device scaling and forces designers to employ bigger devices with more predictable parameters, which pushes the technology back to larger geometries.

The random effects can and often are spatially correlated. This can be used to match the parameters of several related devices, something that is particularly important in analog circuits. Spatial correlation is only significant for relatively small on-chip areas, as demonstrated in [107]. The authors studied spatial correlations of ring oscillator frequencies located within a $64\mu m \times 100\mu m$ tile. They found measurable correlation only within eight ring oscillator radius, which predicts no significant correlations in random PVT variations between the processor cores.

Dynamic variations stem from system architecture, layout, circuit operation and depend on its activity, which is driven by application behavior and resource utilization. These variations can be broken down into reversible and irreversible. *Reversible* effects include temperature/voltage hotspots, self-heating, supply voltage (IR drops) and temperature fluctuations. For example, a multi-core system where only a subset of processors are actively utilized has temperature and voltage hot-spots that contribute to a performance gradient across the cores with otherwise nominal performance. The hot-spots can accelerate uneven circuit aging and contribute to *irreversible* effects. These include Negative Bias Temperature Instability (NBTI), hot carrier effects, electromigration, and dielectric breakdown, all of which degrade transistor and interconnect performance and might eventually result in a permanent fault.

Compensating for the dynamic effects involves a combination of design and run-time techniques. A designer can "distribute" the computation load through the architecture and layout techniques that minimize the performance gradients between the highly and lightly utilized areas of the die. The run-time environment can further minimize dynamic performance variation impact through load balancing.

Systematic and random process variations together with dynamic voltage and temperature variations create a complex circuit performance profile. The interrelationships between process, voltage, temperature, leakage current and the dissipated energy is illustrated on Figure 2.2. Temperature variations stem from different activity factors among cores, functional units, from different circuit structures, and from nonuniform surface of the thermal interface material (TIM) that bonds the chip to its package. Voltage variations stem from IR drops that result from non-ideal voltage distribution, which in turn are exacerbated by activity-dependent IR drops. These are exacerbated by temperature-dependent leakage-current variations (*i.e.*, varying the I term) or switching activity that causes voltage drops due to circuit inductance and possibly insufficient decoupling capacitance. These three variation sources exhibit a number of feedback loops. Process variations affect leakage, which affects both voltage and temperature. Voltage affects energy dissipation and the temperature, which then affects leakage [67].

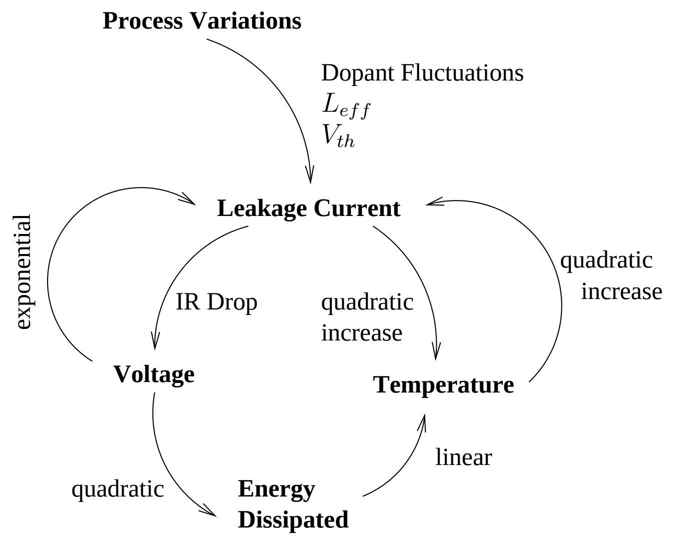


Figure 2.2: The relationship between process, voltage and temperature variations and their impact on circuit performance.

In the modern semiconductor process, the transistor performance is predominantly affected by variations [27], and the impact grows as feature sizes decrease. The interconnect, however, remains largely unaffected by variations. This is particularly true for longer interconnect traces because they are several process generations behind the transistor. However, long on-chip connections and buses are buffered at regular intervals. Since the delay of the optimally buffered interconnect is not dominated by its RC delay, the overall bus delay variations will be largely determined by the buffer variations [112].

Process Step	Physical Effect	Device	Circuit	Architecture
CMP, Etching, Ion Implantation, Lithography, Diffusion, ALD, PVD, Cleaning	*Systematic:* Line Edge Roughness (LER)	V_{th}, W_{eff}, L_{eff}, t_{ox} variations	Timing, Leakage, Noise Margins, Dynamic Energy	Functional (Bit-Error Rate, Faults)
	Stochastic: LER, Random Dopant Fluctuations, Impurities			Parametric (latency, throughput, power consumption)

Table 2.1: Semiconductor process steps and physical effects manifest themselves as functional and parametric variations on multiple levels.

Design Level	Examples of techniques
Process	high-k dielectric, OPC
Layout	transistor size matching, DFM rules
Circuit	adaptation (AFS, AVS, ABB) combined with ECC
Architecture	delay-insensitive (asynchronous) communication, module redundancy, checkers
System	run-time application mapping, routing, reconfiguration, checkpointing

Table 2.2: Techniques to limit faults and PVT variations on every design level.

2.1.2 Minimizing Variations

PVT variations manifest themselves as different effects, parameters and metrics on every system level (Table 2.1). The methods that limit their impact must be applied vertically throughout the design process as well (Table 2.2). As the metrics across the levels are related, the impact of various methods often overlaps, which creates redundancy and wastes resources due to "over-engineering." For example, large layout margin and asynchronous communication discipline both compensate for PVT variations and both incur implementation overhead. Which is more effective? If one is applied, can the designers proportionally scale down the impact of the other? Unfortunately, there is no simple way to trade off the impact and costs of these techniques. This work shows that the extra effort at the system level with dynamic mapping and routing can reduce the margins and costs at the lower implementation levels. The following are some example techniques that compensate for PVT variations on different design levels. No one level can entirely compensate for all PVT effects and certainly cannot do it in a cost-effective manner.

Layout: DFM techniques provide extra margins for yield and reduce process variations, since many of the stochastic effects such as dopant fluctuations are anti-correlated with feature sizes. For examples of layout techniques, [106] analyzed spatial and proximity intra-field gate CD (critical dimension) variability. The authors demonstrated that simultaneously correcting for spatial and proximity effects, which is a result of inadequate mask resolution, yields in 14% ideal circuit speed improvement. This gain is comparable to what we obtained from adaptive routing (Chapter 7). This example shows that a highly complex characterization and correction in a particular process node (the authors used 180nm) may not even compare with the potential improvements from a less costly technique. Note, however, that dynamic mapping and routing compensates for a wide range of performance variation effects, and not only gate CD variability.

Circuits: PVT variations, limits on the supply voltage scaling, and power dissipation have mostly eliminated logic styles other than the static ones. Static logic offers strong noise immunity with complimentary active pull up and pull down networks. The critical path length has also been affected. Prior to 180nm process, the main approach to improve circuit clock frequency was increasingly deeper pipelining, which resulted in short critical paths with significant delay variations and large design margins even in older processes. Coincidentally, [127] showed that short critical paths were also far from the power-performance optimal logic depth of 18 FO4 delays (15 logic + 3 latch). What initially was a power limit trend, had a dramatic impact on PVT variations. Since longer combinational paths tend to “average” gate delays, they result in narrower circuit performance variations (Section 2.2).

Adaptive body biasing (ABB) and adaptive supply voltage (ASV), can be applied to most circuits and have a potential of reducing the impact of process variations by more than 2x [59]. These techniques are mostly orthogonal to the circuit type because they simply realign device parameters from different dice to compensate for die-to-die and wafer-to-wafer variations.

Techniques, such as error correction, can be used efficiently only in some domains (*e.g.* memories and communication) to dynamically trade off power consumption for error rate [109]. For example, [137] discusses a Dynamic Voltage Swing Scaling (DVSS) for communication in Network-on-Chip. The voltage swing is adjusted to minimize energy consumption while maintaining very low transmission error rate, which results in up to 42% energy savings. This is a good example to demonstrate that the system implementation level where to apply ECC techniques must be carefully selected to yield a better solution. In [137], DVSS operates transparently and independently with respect to the rest of the system. This simplifies its implementation as compared to the case where upper layer firmware or the operating system controls the voltage swing. At the same time, the DVSS scheme implements retransmission, which consumes power and complicates the design. A better approach might be for delivery mechanisms, including retransmission, to be a part of a NoC router logic and/or the run-time QoS framework.

This allows the environment to select the optimal operating network regime given its load, power and performance requirements.

Architecture: This research strives to demonstrate that architecture aware mechanisms supported by the run-time system can compensate most flexibly and efficiently for the performance impact of PVT variations (Chapter 3). The architecture provides control knobs and sensors to the run-time layer that can adjust the system operating regimes for optimal utilization at lowest power. For example, [95] implements small test oscillators on across the chip to monitor temperature and voltage supply to the area and control the operating regime for the adjacent modules. This circuit characterizes the performance profile at and around its site. Due to spatial correlations, the test circuit experiences the same process variation effect as its neighbors, and thus it can adjust within its own operating range (Figure 2.7) autonomously.

2.1.3 Problems with Current Approaches

Until recently, the success of VLSI CAD came from the abstractions that separated concerns between the architects, RTL designers, circuit designers, and layout engineers. Architects deal with functional modules, RTL designers — with gates and registers, circuit designers — transistors and wires; and layout engineers — rectangles. These clean abstractions are no longer sufficient to produce high performance, cost effective VLSI because they limit cross-layer visibility and preclude effective and consistent methodology to trade off reliability and yield for power, area, performance.

The divide between electronic design and DFM rules is particularly striking. Theoretically, the trade off between DFM yield improvement *vs* lost performance and power can be quantified for individual layout rules; in practice, rule combinations present a challenge. Foundries typically specify safe guidelines to produce transistors and interconnect segments that match the properties in their SPICE models optimized for several performance corners, *e.g.* low power slow transistors or high performance ones. The presented guidelines are deceptively simple, and their actual effectiveness changes with process maturity and the design type. Since the rules apply to layout sizes and shapes, they preclude a meaningful connection to and understanding by the circuit designers who think about transistors, gates and modules rather than layout rectangles. Designers essentially apply all rules without any quantification of their effectiveness, their criticality, and their cost [57]. The outcome of this uncertainty is lost performance and power due to transistor and wire sizing, spacing, and overly safe voltage supply margins.

Table 2.2 provides examples of resiliency techniques that can be applied independently on various design levels. These point solutions range from brute force dual modular redundancy [81] to invariant checkers in a processor core [97]. A cross-layer optimization strategy is required for fault and variation resilience, a strategy that spans from the lowest level of process and device engineering to the upper level of system architecture. [28] presents components of this strategy. It investigates simultaneous circuit yield and energy optimization, and identifies several key parameters (W, V_{dd} and $\frac{V_{dd}}{V_{th}}$) that most critically affect the yield. This can be a part of a system-wide strategy, where designers identify critical parameters that maximize yield (*e.g.* supply to threshold voltage ratio) and provide control knobs (*e.g.* Adaptive Voltage Scaling) to run-time system. These knobs allow to dynamically select the most appropriate operating point for a particular process corner that affects the die and its sub-components.

2.2 Variation Modeling

2.2.1 FMAX Model and Simulation Infrastructure

Due to a great number and variety of physical processes and variations that affect multi-core system performance, an analytic model has a limited practical, usable impact. A small circuit with a same logic type (*e.g.* datapath, memory, or random logic) can be modeled to gain an insight into its performance profile given a process variation model from the fabrication facility. Modeling a complete system comprising different logic blocks, including memories and compute elements, is complicated and does not yield very useful analytical results particularly when dynamic effects and specifics of the on-die layout are added to the model. A useful, but costly method to evaluate multi-core system performance is with Monte Carlo simulations, since the stochastic nature of this process naturally identifies both the expected performance and its variance, as well as critical corner cases.

Transistor level Monte Carlo simulations are too costly and impractical, and they require accurate and complete models. This work uses router level Monte Carlo simulations, instead. The models are parametrized with router performance metrics, such as packet transmission latency, router and channel throughput. This high-level analytical framework models NoC router parameters using transistor parameters and their PVT variations. This framework is based on FMAX [22] and yields a first order router performance model. The presentation below is an overview rather than a full usable model. Its quality of results depends on the semiconductor device parameters and the accuracy of the abstract router model, both of which are dependent on a concrete NoC implementation and cannot be abstracted beyond what we have done below.

Consider a transistor process variation model characterized by four parameters: width W_{eff}, length L_{eff}, threshold voltage V_{th}, and oxide thickness t_{ox}. These variables are *random* and possibly correlated, as the case is with the threshold voltage shown in Equation 2.1. Based on these parameters, one can compute the expected gate delay in a chosen circuits using advanced device models [25] or very simple first order models [112] sufficient for our qualitative model:

$$\begin{aligned} I_{dsat} &= k' \frac{W_{eff}}{L_{eff}} \left((V_{dd} - V_{th})\, V_{dsat} - \frac{V_{dsat}}{2} \right) && (2.2) \\ R_{eq} &= \frac{3}{4} \frac{V_{dd}}{I_{dsat}} \left(1 - \frac{7}{9} \lambda V_{dd} \right) && (2.3) \\ C_L &= (1+\alpha) W_{eff} L_{eff} \frac{\epsilon_{ox}}{t_{ox}} && (2.4) \\ T_{gate} &= 0.69 R_{eq} C_L && (2.5) \end{aligned}$$

Equations 2.2 and 2.3 compute transistor's equivalent resistance in the saturation region, which translates into gate's pull down network. Equation 2.4 estimates the load capacitance as gate capacitance of the subsequent gate, where $\alpha = \frac{W_{PMOS}}{W_{NMOS}}$ is the sizing width ratio between the PMOS and NMOS transistors in a gate (typically $\alpha > 2$ for an inverter). The expression for gate delay in Equation 2.5 treats gate switching as discharge through an RC network. Since the input parameters V_{th}, W_{eff}, L_{eff}, and t_{ox} are random variables, so is the delay T_{gate}.

To obtain the circuit performance, a critical path delay or maximum operating frequency, we define an abstract circuit model based on FMAX [22]. An abstract circuit is described by two parameters: the length L_{cp} and the number of *near-critical* circuit paths N_{cp}. A nominal circuit has one longest combinational path, but with PVT variations any of the near-critical paths could potentially become the longest and thus critical:

- L_{cp} is the average length of the *near-critical* paths that could influence the maximum frequency. For this model, L_{cp} represents a path length in gates, *e.g.* a fanout of 4 inverter delays. If gate delay T_{gate} is a random variable described with the expected and standard deviation values $[\mu(T_{gate}), \sigma(T_{gate})]$, then the critical path delay is the sum of L_{cp} gate delays:

  $$T_{cp} = L_{cp}T_{gate} = [\mu(T_{gate}) \times L_{cp}, f(\sigma(T_{gate}))] \tag{2.6}$$

  where T_{cp} is the probability density function (PDF), *i.e.* the distribution of critical path delays. It is described with the expected value and the standard deviation shown as a tuple. The standard deviation of T_{cp}, $f(\sigma(T_{gate}))$, is a function of the gate delay distribution and spatial parametric correlations between adjacent devices. In the following, we assume normally distributed gate delays T_{gate}, but any symmetric distribution yields equivalent qualitative results.

  If T_{gate} variation is systematic and/or spatially correlated, the path variance is the sum of gate variances. L_{cp} has no impact on the standard deviation of the critical path [22]:

$$\frac{\sigma(T_{cp})}{\mu(T_{cp})} = \frac{L_{cp} \times \sigma(T_{gate})}{L_{cp} \times \mu(T_{gate})} = \frac{\sigma(T_{gate})}{\mu(T_{gate})} \tag{2.7}$$

If systematic effects continue to have a dominant impact on process variations, then Core-to-Core performance variations are going to be determined primarily by these effects.

In contrast, should the gate delay variations be stochastic and mostly spatially uncorrelated, longer critical paths will average out $\sigma(T_{gate})$ and diminish its impact. Normal distribution is assumed here to show a closed form solution [15]:

$$\frac{\sigma(T_{cp})}{\mu(T_{cp})} = \frac{\sqrt{L_{cp}} \times \sigma(T_{gate})}{L_{cp} \times \mu(T_{gate})} = \frac{1}{\sqrt{L_{cp}}} \times \frac{\sigma(T_{gate})}{\mu(T_{gate})} \tag{2.8}$$

Larger L_{cp} increases averaging effect for uncorrelated variations and reduces path delay variability. Figure 2.3 shows that the actual circuits exhibit some degree of spatial correlation in gate delays [22].

- N_{cp} is the number of *near-critical* combinational paths in the circuit. The longest of N_{cp} paths determines the maximum circuit clock frequency. Given the distribution $T_{cp}(x)$ of the critical path delays, the circuit delay distribu-

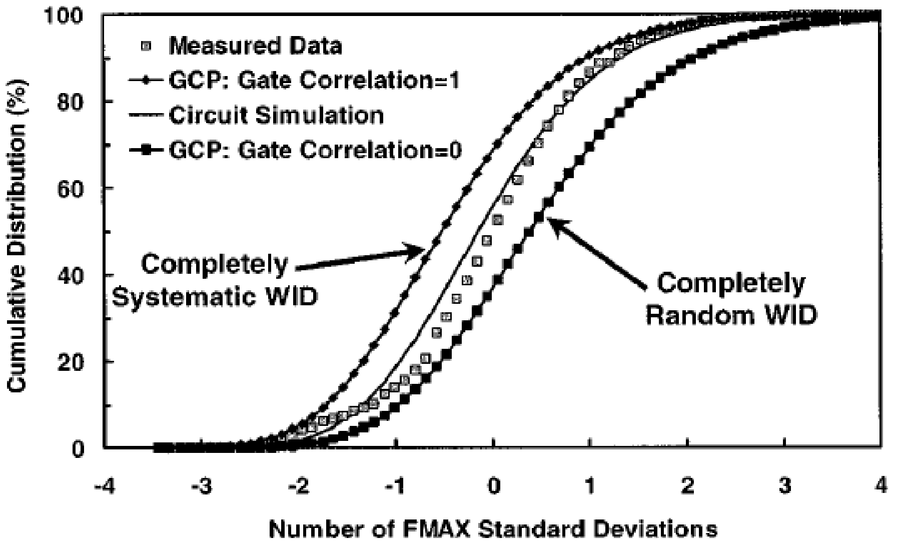


Figure 2.3: Real gate delay exhibits some spatial correlation [22].

tion is:

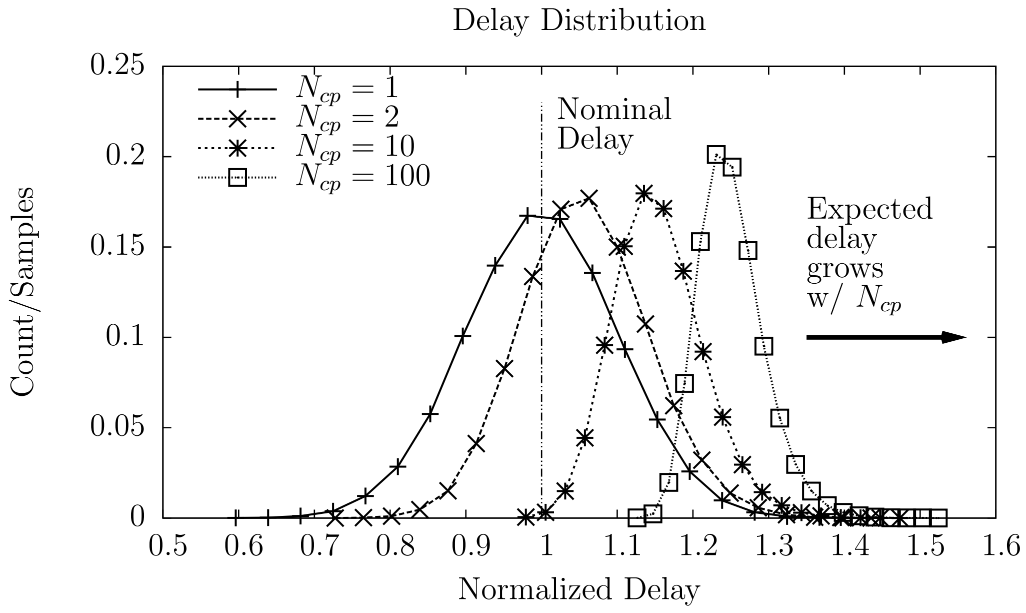
$$T_{circuit}(t) = N_{cp}\left(\int_0^t T_{cp}(x)dx\right)^{Ncp-1} T_{cp}(t) \tag{2.9}$$

which is derived from cumulative density function (CDF) [22]. Larger N_{cp} increases the expected critical path delay because the probability of an outlier path with a delay close to $\mu(T_{cp}) + 3\sigma(T_{cp})$ is greater as shown on Figure 2.4(a). Unfortunately, even with normally distributed critical path delay, no closed form expression for the expected circuit delay $\mu(T_{circuit})$ and its standard deviation $\sigma(T_{circuit})$ is available. Their evaluation must be performed numerically.

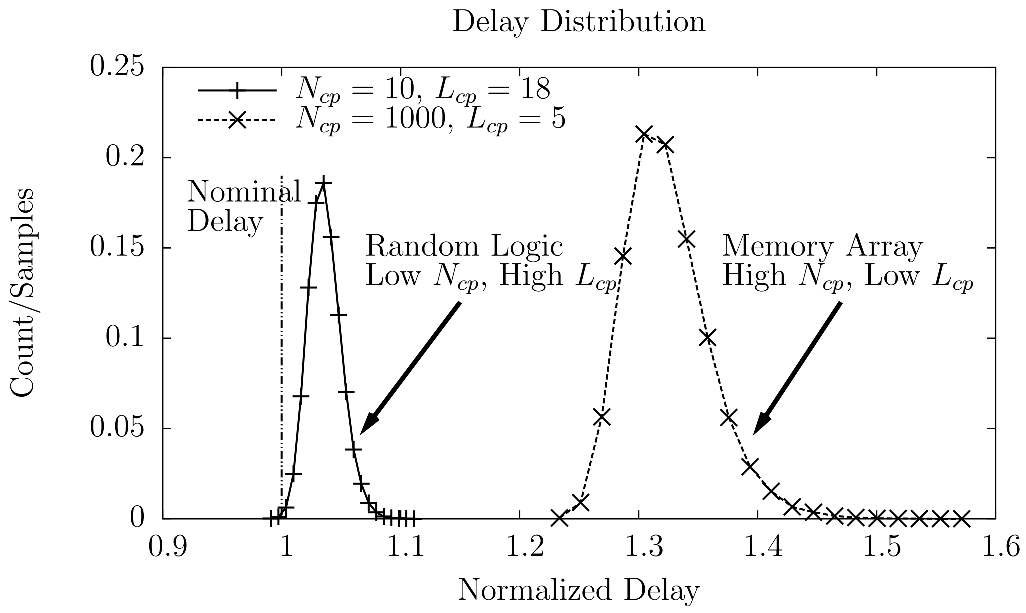
Table 2.3 shows two important circuit types — random logic and memory array — with a qualitative description of their parameters. Figure 2.4(b) illustrates circuit delay probability density functions, which assumed normalized gate delay distribution $T_{gate} = (\mu = 1, \sigma = 0.1)$. As expected, random logic benefits from averaging effect along its few long critical paths, which lowers the expected path delay and narrows the path delay variance. In contrast, the critical paths in a memory array comprise mostly interconnect, which is not significantly affected by the variations, and a few logic gates, which are affected. Low L_{cp} reduces averaging. However, a high N_{cp}, which corresponds to the number of columns and rows in the memory array, significantly increases the probability of an outlier and pushes the expected critical circuit delay higher.

In a typical design, the cache array will dominate overall processor performance, making the case for decoupling of the two subsystems with an asynchronous interface. Chapter 3 argues for a multi-core GALS architecture with synchronous core/router islands as a method to expose core-to-core PVT variations. Further decoupling of pipeline stages within a core has also been demonstrated to yield higher system performance and power savings as compared to the globally synchronous design even in cases without PVT variations [92].

To obtain a system performance model, FMAX can produce the critical path delay distributions for each subsystem within a processor core and an on-chip



(a) Effect of N_{cp} on circuit performance.



(b) Performance Distribution for randome logic *vs* memory arrays.

Figure 2.4: Impact of L_{cp} and N_{cp} on circuit performance distribution.

Logic type	N_{cp}	L_{cp}
Random Logic	Small: 10s – 100s	Tens of gates
Memory array	Large: SRAM bits × read ports	Few (mostly interconnect)

Table 2.3: Summary of N_{cp} and L_{cp} parameters in the FMAX model [68].

network router characterized by L_{cp} and N_{cp}. This work focuses on the network. Based on the delay distributions, the Monte Carlo method synthesizes NoC instances annotated with performance parameters: router throughput and latency. Instead of treating critical path delay as a distribution of *absolute* values induced by the underlying transistor performance, we decompose the delay into the nominal value with "deltas" for each variation type:

$$T_{cp} = T_{cp,nom} + \Delta T_{cp,stoch} + \Delta T_{cp,sys} + \Delta_{Temp} + \Delta_V \tag{2.10}$$

Each "delta" mimics the variance and "shape" of a corresponding distribution, but sets the expected value at 0. $\Delta T_{cp,stoch}$ represents contribution from correlated and uncorrelated stochastic effects; $\Delta T_{cp,sys}$ represents the contributions from systematic effects; Δ_{Temp} and Δ_V — from dynamic on-die temperature and voltage. Notice, that an important model component is missing: on-die circuit location. Systematic effects and dynamic effects are location dependent, and stochastic effects are *not* independent of the systematic ones, as the following relationship

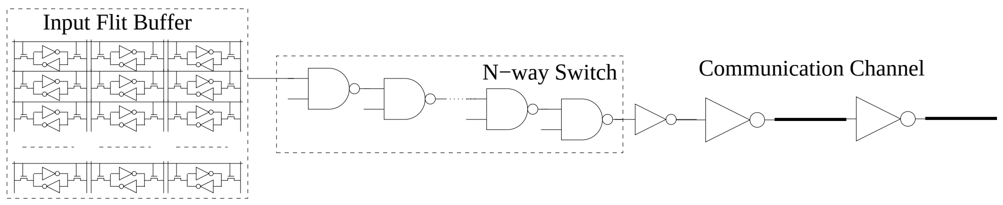


Figure 2.5: Simple first order router model

demonstrates:

$$\sigma_{V_{th}} \sim \frac{1}{\sqrt{W_{eff} L_{eff}}} \tag{2.11}$$

Although, the threshold voltage is random, a result of stochastic nature of ion implantation, its distribution variance depends on the actual device size, which is a result of systematic and stochastic effects. Addition of on-die locations (x, y) creates a complete model that can be simulated at multiple levels: devices, router components, and router performance:

$$T_{cp}(x,y) = T_{cp,nom}(x,y) + \Delta T_{cp,stoch}(x,y) + \Delta T_{cp,sys}(x,y) + \Delta_{Temp}(x,y) + \Delta_{V}(x,y) \tag{2.12}$$

In this work, an on-die location (x, y) refers to an area containing a sub-module such as router input FIFO, rather than a transistor or an interconnect trace.

Figure 2.5 shows a simple first order router model, comprising three key components in the router critical path: input flit buffer, a switch, and a communication channel to the adjacent router. The simulation framework can compute L_{cp} and N_{cp} parameters based on a NoC router implementation:

- Input Flit Buffer performance is determined by its size (B flits[1]), and the system flit size (w bits). Assuming that each flit buffer is an independent block, implemented in a single SRAM array of size $B \times w$, the FMAX model parameters are defined as follows. $N_{cp} = \max(B, w)$ and $L_{cp} = \log(B) + T_{SA}$ comprises row decoder delay and sense amplifier delay required to capture the values from the array. Note, that we omit the array access time as it mostly interconnect and unaffected by PVT variations relative to transistors. For an implementation that uses registers in place of an SRAM array in order to achieve smaller input buffer sizes and reduced power consumption [66], the definitions of L_{cp} and N_{cp} also hold.

- The switch can be defined by its connectivity and degree, the number of inputs and outputs. For simplicity, assume a fully connected crossbar with N inputs, implemented as a fan-in tree of multiplexers. $L_{cp} = \log(N)$ is the tree height, and $N_{cp} = N^2$ represents the total number of paths in a crossbar.

[1]A flit is a unit of flow control in a network, typically a word or a double-word

- The communication channel is implemented as a repeated interconnect bus of width w and length L. An important factor differentiated this router component from input flit buffers and the switch is that a communication channel spans a large area defined by the source and sink router locations $(x_0, y_0) - -(x_1, y_1)$. The on-die distance between the routers and the underlying technology determines the optimal repeater (or pipeline register) distance and thus L_{cp} parameter. N_{cp} is defined simply as w in a basic unidirectional bus. The repeater locations along the bus determine their specific systematic "deltas."

Based on the FMAX parameters for each router subsystem, compute router throughput and latency, which depend on implementation details such as degree of pipelining of the router. Assume that each of the subsystems is a separate pipeline stage and the inter-router channel is *not* pipelined, then the throughput W_{rout} and latency L_{rout} can be expressed as:

$$W_{rout} = \max(T_{cp,buf}, T_{cp,sw}, T_{cp,ch})^{-1} \tag{2.13}$$

$$L_{rout} = T_{cp,buf} + T_{cp,sw} + T_{cp,ch} \tag{2.14}$$

where router position (x, y) have been omitted for simplicity and clarity.

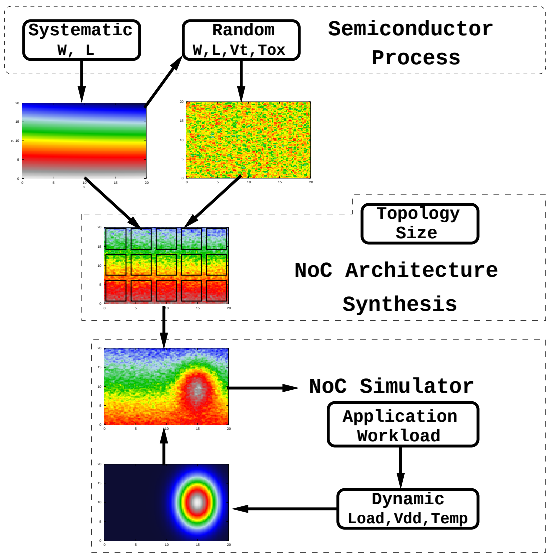


Figure 2.6: NoC simulation framework forms a part of variation-aware architecture exploration environment.

Figure 2.6 illustrates the structure of the Network on Chip simulation framework. The semiconductor process provides the device models: transistor dimensions W and L, the threshold voltage V_{th} and oxide thickness t_{ox} as spatially dependent distributions. This level of detail is required to compute the nominal circuit performance $T_{cp,nom}$, stochastic $\Delta T_{cp,stoch}$ and systematic $\Delta T_{cp,sys}$. A multi-core architecture instance defined by the number of cores, network topology, and other parameters discussed in Chapter 3 is realized (placed) on the manufactured die profile that includes stochastic and systematic variation contributions. This gives each router and channel a performance profile (Figure 2.7) depending on its on-die location.

A network on chip simulator enables dynamic performance measurements for a collection of application workloads (Chapter 5). The simulator captures resource utilization and activity statistics for each router and channel, which can be translated into power consumption by individual network components, uneven voltage distribution across the die and changes in on-die temperature gradient. These dynamic effects Δ_{Temp} and Δ_V impact core and router throughput and affect overall system performance, forming a cyclical dependency in the simulator framework. This work does not directly study the effects of dynamic VT variations on the application performance, since the details of the plausible model are heavily dependent on a complete system implementation. Instead, the assumption is that the system temperature and voltage attain a steady state, and each core/router tile settles in its operating regime for the length of application execution.

The experimental results depend directly on the semiconductor variation model quality and accuracy. If a sufficiently accurate model is not available as is the case in this work, it is possible to perform simulations by modeling PVT variations as the core/router performance directly (Chapter 7). This serves two main goals. First, it permits to examine foreseeable scenarios in the future process generations and come up with prescriptive approaches to deal with variations. Second, it enables simulations independent of a specific process or its maturity, which typically involves proprietary information. Instead, with NoC level variation models, it is simpler to isolate different router components' performance distributions and understand their impact on the overall system performance. By analyzing the sensitivity of overall system performance to the performance variance of individual components, one can identify the subsystems where designers should place most effort to compensate for PVT variations *vs* others where the variations can be compensated more effectively by an adaptive routing scheme.

2.2.2 Variations and Multi-core

What is the impact of PVT variations on a multi-core die? What does the die performance landscape look like? Two multi-core architecture trends have emerged so far. The first follows a conventional track of highly complex, super scalar, out-of-order execution processor cores that optimize single thread performance with large caches, speculation and out-of-order execution: Intel Itanium [128], Merom [116], and AMD Opteron [43]. The second track is a push toward simpler, in order, multi-threaded cores designed to tolerate high unpredictable memory access latencies common on a multi-core die with limited off-chip bandwidth: Sun Niagara 2 [102], Cisco Metro [51], and Intel Teraflop 80-core prototype [135].

In both cases, power-performance optimization and increasingly heavy influence of PVT variations forced the designers to lengthen critical paths and reduce clock frequencies. With longer combinational paths, within-core stochastic variations partially average out and reduce core performance variance (Equation 2.8). However, within die, chip-to-chip and wafer-to-wafer systematic effects become the dominant factors that widen the *variance* of core performance [68]. If each core is a synchronous island, connected to its peers with an asynchronous interface, the die becomes a collection of cores with heterogeneous performance as shown in Figure 2.7. Each core has an *operating region* defined by its local process variations, *i.e.* the distributions of the critical dimension size, oxide thickness, and threshold voltage. The dynamic supply voltage scaling and body biasing enable the designer and the run-time system to adjust the core performance within its functional operating region. Assuming that stochastic variations are averaged in long critical paths, the spread of core-to-core variations (C2C) will be mostly dictated by on-chip systematic effects.

For designers, the implementation difficulty arises from three factors. First, as they push for parallelism rather than frequency scaling, an in-order core may have longer critical paths (L_{cp}), but also a larger number of *near-critical* paths (N_{cp}). Any one of those paths can become critical with PVT variations and affect the clock frequency, and as the number of these paths increases, so does the chance of an outlier pushing the clock frequency down. Second, even with gate delay averaging in longer critical paths, the impact is limited and depends on spatial correlation as shown in Equations 2.7 and 2.8. As transistor variations grow, the performance impact from stochastic variations cannot be tightly controlled. Third, although the exact breakdown is not known for the future semiconductor processes, some sources suggest diminishing impact of systematic effects [107] and dramatic increase in stochastic PVT variations [134]. This is consistent with our intuition that process controllability and accuracy decreases with smaller device geometries that lead to growing stochastic effects. To complicate the matters, in the future processes performance variations cannot be strongly correlated with any one device parameter but is related to all simultaneously. This results in a truly unpredictable and uncertain system components [41].

A more scalable approach is to treat a multi-core die affected by variations as a collection of black boxes that abstract the underlying variation sources as observable performance. Although modeling and characterizing the impact of variations on manufactured dice can highlight the relevant trends, it still leaves designers with a heterogeneous platform that is best managed dynamically with task and communication mapping.

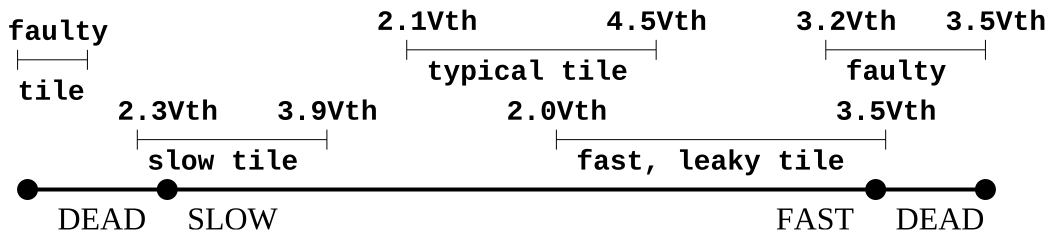


Figure 2.7: A sampling of operating regions for different types of on-die tiles.

2.3 Summary

With semiconductor devices scaling to ever smaller dimensions, a designer's ability to accurately control device parameters diminishes. The performance degradation caused by inter-related Process Voltage Temperature variations is predicted to grow further in the future processes. The current trends in the sub-32nm process point to the increasing influence of within die, die-to-die and wafer-to-wafer stochastic and systematic variations, which result in a complex landscape of core-to-core performance variations.

The damaging impact of the PVT variations can and must be addressed on every system level: from the layout to the architecture. Low level layout and circuit techniques can be efficient and effective if variations are well characterized. However, this characterization is increasingly difficult in an environment with smaller critical dimensions and immature semiconductor processes. More importantly, even with accurate characterization, most low level techniques reduce to large guard-bands and margins rather than focused approaches. The problem stems from the fact that architects and circuit designers do not have tools to simultaneously optimize power and performance against parametric PVT variations. This is because of a large abstraction gap between their system representation and the semiconductor chemical-physical processes. Six sigma DFM guard-bands applied on multiple system levels result in significant wasted area and performance, and motivate us to look for a different solution.

Chapter 3

Multi-core Architecture for Heterogeneous Performance

Many researchers have proposed and hailed multi-core architectures with Voltage Frequency Islands (VFIs) for their power efficiency [92, 30, 103]. This chapter identifies the critical features of these architectures as they apply to fault and PVT variation resilience.

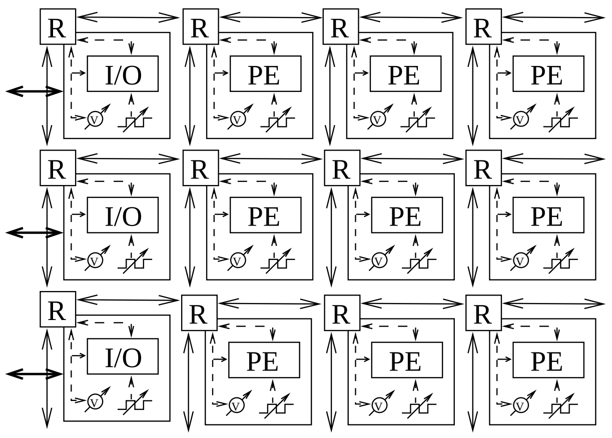
Alain Martin has for a long time championed asynchronous circuits because of, among other things, their natural tolerance for PVT variations [94] and soft-error robustness [73]. These circuits incur high area overhead due to additional signal coding as compared to their synchronous counterparts, making them prohibitively expensive on the gate level. At a coarser level of inter-core communication, the asynchronous handshaking overhead can be relatively small and acceptable.

As described in Chapter 2, Process Voltage Temperature variations create a die with a complex performance profile. In the spirit of asynchronous logic that delivers *average* rather than the worst case delay, the most natural implementation of a multi-core system on a die affected by the variations is to give every core its own clock domain. Each core then can run it at its optimal regime within its own operating range, dictated by the core's on-die location, PVT variations and the application load. With appropriate buffering, this globally asynchronous implementation enables communication between these heterochronous[1] cores [112]. A stylized Network-on-Chip thus evolves naturally out of such Globally Asynchronous Locally Synchronous (GALS) design, as it provides the required buffering and flow control between the communicating processor cores that perform rate matching [37]. With its point-to-point links, a NoC is a scalable alternative to buses and crossbars, which is critical as the number of on-die cores grows.

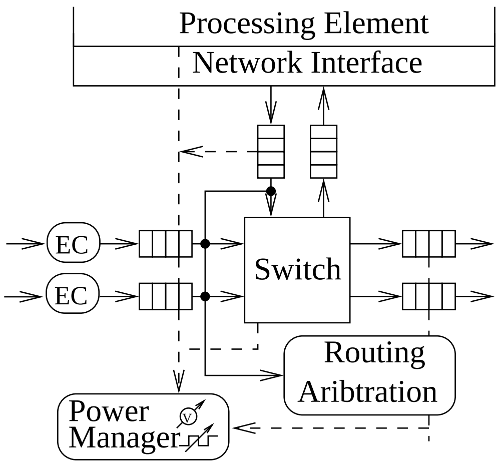
3.1 Compute and Router tile

Multi-core systems offer a new opportunity to benefit from asynchronous design, by capitalizing on the best of both worlds of GALS architectures. Small processor cores can be implemented efficiently in their own local clock domains, benefiting from exiting CAD tools and implementations. Limiting the on-die range of clock distribution network to a small core, helps with design performance scaling because local interconnect delay scales approximately at the same rate

[1]Heterochornous cores have an unknown clock relationship.



(a) Power/CLK island architecture



(b) Tile Architecture (router focus)

Figure 3.1: A multicore tiled architecture envisioned in this work.

as a transistor [65]. However, global inter-core communication should decouple the cores, be delay insensitive, and reap the benefits of variations tolerance and power efficiency [7]. For example, [17] showed that a globally asynchronous implementation decreases communication latency by 25% relative to a synchronous implementation. The authors so far have only considered a small scale 5-core design, but the advantages of GALS on a larger multi-core design will only increase.

The appearance of multi-core architectures was a natural response to the limits in clock frequency and power dissipation scaling as semiconductor processes moved to increasingly smaller device geometries [65]. In addition, limited Instruction Level Parallelism (ILP) in a typical sequential program makes the promise of the continuing single core performance scaling infeasible. Multi-core architectures hold enormous potential for massively parallel applications, but present a considerable number of challenges for the architects because the system performance can no longer be neatly described by a handful of execution units and average cache performance. Instead, it now depends on a number of factors such as network performance and effective throughput to off-chip memory and I/O that many cores could simultaneously be vying for.

Fortunately, with respect to resilience to PVT variations, multi-core provides huge opportunities. We focus on a multi-core architecture comprising regular and redundant compute/communication tiles organized in a grid. Regularity enables efficient and complete system abstraction: a tile represents a task that communicates with its peers. Redundancy provides a model for fault and PVT variation tolerance and core virtualization. Together, regularity and redundancy allow the system architect to move away from point solutions toward a systematic approach to resilience, assisted and automated by VLSI CAD tools, compilers and the run-time system. With regularity and redundancy, a tile can be abstracted with a few relatively simple power and performance metrics such as compute or routing throughput and response time. Tile utilization can be dynamically optimized without concern for its PVT corner or other factors such as distance to off-chip memory that can affect its performance. Finally, the tile can be virtualized through support of task check-pointing and migration, assuming a configurable communication fabric.

From the hardware implementation point of view, redundancy and regularity reduce manufacturing and verification effort because designers produce several types of tiles (*e.g.* compute and I/O) and replicate them across the die. Regular die structure simplifies global power and reference clock distribution as well.

The key tile components include a processing element (core) and an interface to the global communication fabric implemented as a network on chip router. I/O blocks, typically located on chip periphery, and any custom hardware accelerator blocks require the same interface to the on-chip network (Figure 3.1(b)). Each tile contains local voltage and clock management circuitry, that can be controlled by firmware, hypervisor or the operating system. For implementation performance and efficiency, it is possible to combine the router with the core to reduce communication latency. This may further simplify router implementation and improve its performance, because handling of the special, infrequent, and corner cases (*e.g.* router reconfiguration or error handling) can be conveniently implemented by the processor core instead of a rarely utilized router circuit. For clarity, this work focuses on the router as a separate module from the core.

The key architectural feature is tolerance of PVT variations and faults across a set of cores, which requires electrical isolation between core/router tiles. This allows the chip manufacturer to turn off faulty tiles after production, and the chip user to shut down the slow or unused tiles dynamically at run-time [141]. In other words, electrical isolation creates core fault containment, which is required to ensure that a die is usable even with some faulty cores. This is the critical feature for core sparing (Chapter 4). Earlier, tile clock domains open the doors for Dynamic Frequency Scaling (DFS), turning tiles into Voltage-Frequency Islands.

With electrical isolation, each core/router tile may offer the following controls (knobs) to the run-time environment: Dynamic Voltage Scaling (DVS), DFS, Power on/off. Due to PVT variations and faults, a multi-core die is as a collection of cores, each with its own performance *range* shown on Figure 3.2. This range represents all performance points where a core functions error-free without timing violations, or excessive static current and power dissipation. The mean value of this range at a particular temperature or voltage is determined by the semiconductor process corner — an aggregation of process variations effects — that impacts the core. The range *width* is determined by Process Voltage Temperature variations, which impose V_{DD} to V_{th} ratio, noise margins and thus limit the performance range navigated by DVS/DFS.

The critical component of this strategy is for cores to expose some metrics that characterize the application task execution performance. The metrics may include fault detection, operating range characterization, performance counters, and flow control.

Fault detectors span a large variety of mechanisms including Error Detection on communication channels [137] and cache banks, Dual Modular redundancy on critical core subsystems [75], and more sophisticated approaches that detect faults as data and control flow micro-architecture invariants [97]. These are designed to signal the presence of soft and hard faults to the run-time environment.

Handling these faults needs to be treated differently from their detection. Traditionally, the micro-architecture was responsible for error correction, such as in the case of ECC protected caches. This work advocates fault handling at a higher level with core sparing for permanent manufacturing or circuit aging faults, and with check point and replay mechanisms for soft errors [53].

Operating range characterization enables the run-time system to introspectively determine the core operating range size. Two implementation approaches are possible. The first one relies on characteristic circuits embedded within the core that adaptively identifies maximum and minimum clock frequencies and voltage range for a current circuit temperature and process corner. Thus, continuously at run-time, this circuit outputs the "bounds" for operating region. This can be used by the run-time system to evaluate the current core/router knob settings *vs* the complete operating range. The run-time system is responsible for setting the knobs such that they are within the valid operating range. The hardware is only required for interlocking to ensure that the software does not drive a core into an unsafe mode of operation. The second approach makes use of performance counters, described below.

Performance counters can characterize run-time performance of critical subsystems such as caches, execution units, and our case the NoC router as well. The counters can be used for auto-tuning, a process of searching an algorithm implementation parameter space for the maximum performance point by measuring the critical events particular to the specific algorithm [38]. For example, if a symmetric multi-threading core has a significantly higher cache miss rate than its peers, relocating more threads onto this core may maximize its overall throughput by better overlapping computation and communication. These tuning optimizations are very computation specific, and a continuous auto-tuning on a multi-core architecture requires cooperation between the applications, its custom task scheduler (if it exists), the operating system, and the hardware. For example, [99] investigates simultaneous design time hardware/software auto-tuning, but this work focuses on the run-time optimization.

For Network-on-chip router performance optimization, the performance counters can capture channel utilization, average throughput, and packet waiting times or input queues occupancy. These metrics allow the router to autonomously optimize its own power /performance efficiency, but the counters can have more impact when viewed collectively by the run-time system. For example, a simple approach to minimize network power consumption is to use DFS/DVS to increase the throughput of the critical congested routers and keep the non-critical routers at lower power levels sufficient to carry the offered traffic. Essentially, the system introspectively maps the tasks and tunes the hardware platform by observing the router throughput. This can be implemented in several ways:

1. Each core/router tile autonomously adjusts its own operating point, based on local (or neighborhood) compute and traffic load measurements. This complex option requires a very careful implementation to avoid network performance degradation and traffic load oscillations due to poor local decision making. At the same time, our adaptive algorithms in Chapter 6 make local autonomous decisions and achieve performance improvements through effective communication load balancing. The role of the run-time system is relegated to task-to-core mapping that establishes inter-core communication flows that NoC routers attempt to accommodate. The main concern in such a distributed dynamical system is the response time, critical for short term bursts and changes in the application traffic pattern.

2. Each core and router provides the performance counters to the operating system that has a global view of the platform and its resource utilization. The OS performs task-to-core mapping and load balancing, adjusting each

core/router operating point to accommodate the expected load, and does so continuously. The hardware only provides the interlocking safeguards to ensure that the OS does not drive it into an unsafe mode of operation.

Flow control, typically associated with communication, is an indirect but a very powerful way to expose core/router performance to the run-time environment. Many computing processes are also designed with implicit flow control. Consider a case where processors pull ready to execute tasks off a logically common work queue (*e.g.* work-stealing) [12]. In a system with cores with heterogeneous performance, faster cores remove more tasks from the queue than the slower cores in the same amount of time, thus naturally balance the computational load. In networking, flow control refers to a synchronization technique that matches the communication rates of the producer and consumer (discussed in Section 3.2). In the case of heterogeneous performance routers and communication channels, inter-router link flow control exposes the performance variations and faults as busy channels and network congestion to adjacent routers. This abstraction has a natural affinity with adaptive routing algorithms. Congestion-aware routing can compensate for static — process variations and faults — and dynamic effects — voltage and temperature induced performance variations — in addition to inherent application communication traffic imbalance.

What are the costs and benefits associated with the proposed multi-clock and supply voltage architecture? Implementation of independent core-level clock and voltage domains suffers from the overhead of interface circuitry such voltage level shifters [61] and asynchronous FIFO buffers [114] that connect heterochronous domains. Dynamic power management has been employed extensively inside [116, 140] and outside processor cores [141], but unfortunately its overhead has not been quantified, but is likely not to be significant on core level. Some academic results indicate that a NoC can reduce power consumption up to 40% with core voltage and clock scaling at the small performance penalty of 10% on a 6×6 array of processors [144]. More research and implementations are required to obtain a definitive answer, since the reported results are not general, but tied to design choices and assumptions.

The main complexity of mixed-clock and voltage designs actually comes additional verification and validation efforts, since the space of operating regimes grows with each extra control knob. In spite of these difficulties, we argue that with increasing impact of PVT variations and faults, reducing the granularity of power and performance management units is the way to extract the expected rather than the worst case performance out of multi-core architectures. For example, according to [30], intelligent task mapping onto an architecture with independent

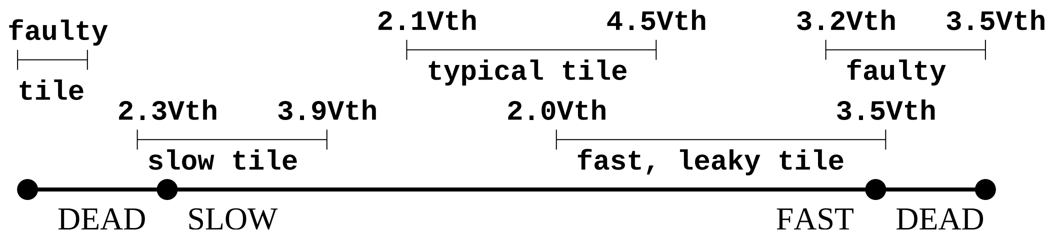


Figure 3.2: A sampling of operating regions for different types of on-die tiles.

core clock domains reduces communication power consumption as much as 50%, effectively without a loss of performance.

As multi-core architectures scale in the number of cores per die, the cores may not remain functionally identical (Table 3.1). A die may include custom hardware functions and domain specific accelerators (*e.g.* cryptography engines [98]), or cores with heterogeneous performance with single or multiple ISAs [82, 110]. Single ISA and multiple implementations is a particularly interesting variant, as it has performance/power benefits over its counterpart with homogeneous performance (single implementation). Dynamically managing homogeneous or heterogeneous cores is fundamentally equivalent. Although mapping and scheduling problems are potentially harder due to restricted choices, the solution is tractable [121]. A compiler provides a single relocatable binary that must be mapped on cores with diverse performance.

Task mapping on functionally homogeneous – performance heterogeneous cores is not fundamentally different than mapping onto a nominally homogeneous ar-

Functionality / Implementation	Homogeneous (Single ISA)	Heterogeneous (Multiple ISAs)
Homogeneous (Single Implementation)	Niagara 2, Merom	N/A
Heterogeneous (Multiple Implementations and/or PVT vars)	[Future Systems]	Cell, DSPs, SOCs

Table 3.1: Functionality and performance dimensions of a multi-core architecture.

ray with heterogeneous performance due to PVT variations. Although we focus on PVT variation and fault tolerance on functionally homogeneous cores, our results would be qualitatively similar for a functionally heterogeneous system. The fundamental difference is choice. Homogeneous cores represent the greatest degree of *redundancy* because the entire die contains the same type of a processing element. Functionally homogeneous systems eliminate the PVT variation bias between cores. This occurs when all cores of same type are located in one parametric variation corner of a die, and thus there is not enough inter-core performance diversity.

Even functionally homogeneous systems contain several key heterogeneous components — off-chip I/O interfaces and communication fabric — whose performance is also affected by PVT variations. These components (the "uncore") may not be carefully considered by the architects, and typically we observe their operation and performance through the core performance. For example, core computational throughput often represents not simply the core's capacity but also application memory access pattern, computation to communication ratio, *etc.* In single-core and small multi-core systems, this abstraction simplifies task mapping and scheduling because an OS does not have to directly consider the "uncore" performance. These simplifications are no longer effective to maximize utilization of a modern many-core architecture because its performance is increasingly affected by the network, the memory, and I/O subsystems, rather than predominantly by the processors. For example, [38] showed that NUMA optimizations in multi-threaded stencil kernels can easily double computational throughput on small scale multi-core systems. The role of the communication fabric in a many core system perhaps eclipses that of the computational elements themselves, and thus we consider NoC architecture next.

3.2 Communication Fabric

3.2.1 Motivation for NoC

Application performance is increasingly tied to the "uncore" performance, where the communication network architecture and performance play the most critical role as systems scale to larger number of cores per die. A globally asynchronous communication fabric combined with core-level clock/voltage domains can maximally expose the PVT induced within die performance variations. A stylized Network on Chip naturally realizes the GA paradigm because it provides on-chip routers connected via direct links at the interface points between these domains. The routers include data *buffers* that match the production and consumption rates between two heterochronous components, which is a requirement for multi-voltage/clock system. The network through its bounded node fan-out retains communication scalability when compared to shared media such as buses and rings, and offers opportunities for *distributed* mapping algorithms to improve fault resilience. Finally, similar to asynchronous logic, GALS network connecting heterogeneous performance cores can with adaptive routing deliver the mean rather than worst case network throughput as demonstrated in Chapter 7.

The general purpose nature of the stylized network does bring some area and power overhead to the system implementation. Power consumption is the dominant concern at the moment. For example, the NoC consumed half of the power in Intel Tera-flop 80 multi-processor [20]. The communication fabric is the place where asynchronous logic can truly shine [94, 10]. For communication circuits that connect distinct rate components, the overhead of traffic encoding and explicit hand-shaking is not significantly different from that in a synchronous implementation but has additional resiliency benefits. For example, the dual-rail signaling, which is used in some delay insensitive designs, offers a common mode noise suppression, low swing and power consumption [145].

The envisioned multi-core architecture creates a dynamic, globally asynchronous environment because of the impact of PVT variations and the dynamic application behavior. This environment naturally creates rate-mismatched boundaries between the core/router tiles, and thus rate matching is the key architectural feature to ensure that heterogeneous cores can co-operate efficiently. Continuous rate-matching requires flow control and buffering on the inter-tile boundaries, making a *packet switched* implementation the most convenient choice. A circuit switched would be more appropriate in a predictable static environment with globally scheduled communication resources. Packet switched networks can emulate circuit switching to improve transmission throughput for long data streams (*e.g.* reduce packet or routing overhead), and support traffic with real time, QoS requirements. This can be accomplished with a bandwidth allocation scheme, typically implemented with a run-time scout packets, or static router priorities [64, 48]. All required hardware mechanisms are already available in the packet switched router: switch, arbitration logic, buffering, and router algorithm logic that can handle special cases if necessary.

3.2.2 NoC Organization

Communication fabric organization is a non-trivial problem because a number of simultaneous concerns must be addressed: connectivity and fault resilience, performance, deadlock and livelock freedom.

Connectivity and **Resilience** are properties of a communication topology that determine its capability to connect arbitrary processor cores to each other and off-chip components. Ideally, a communication fabric should not have a single point of failure that can disconnect the network. A resilient network provides for the ability to send traffic around nodes that have failed or under performing and ensure graceful performance degradation.

Performance is generally described by two metrics: average packet latency and saturation bandwidth for a particular communication pattern and routing algorithm. Section 6.1 offers a more detailed discussion of network performance, but here we illustrate the performance metrics with a simple example on Figure 3.3. A three task graph annotated with communication bandwidth requirements, shown on Figure 3.3(a), is mapped onto a two dimensional mesh with link capacity of 6 Gb/s. Figure 3.3(b) illustrates the communication traffic flows between the processing tiles running Dimension Order routing algorithm, which moves the traffic in the X direction first and then in the Y direction. The flow $A \rightarrow C$, which demands 10 Gb/s, saturates the network link at 6 Gb/s. Here, the Dimension Order router only supports the total cross-section bandwidth of 7 Gb/s. The average packet latency, l_{DO}, measured in network hops can be computed as the weighted sum:

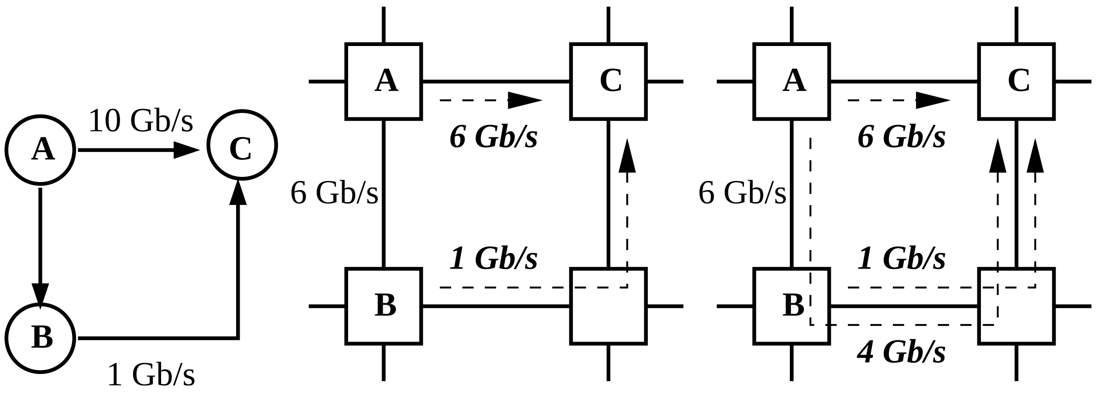
$$l_{DO} = \frac{6 \text{ Gb/s } \times 1 \text{ hop } + 1 \text{ Gb/s } \times 2 \text{ hops}}{7 \text{ Gb/s}} = 1.14 \text{ hops} \tag{3.1}$$

In contrast, when an adaptive routing algorithm is used for the same task-to-core mapping, the network can carry the complete 11 Gb/s of bandwidth demanded by the application (Figure 3.3(c)). Notice, that the adaptive algorithm is non-minimal, in the sense that it pushes 4 Gb/s of A-C traffic through a longer path, which results in a higher average packet latency, l_A:

$$l_A = \frac{6 \text{ Gb/s } \times 1 \text{ hop } + 4 \text{ Gb/s } \times 3 \text{ hops} + 1 \text{ Gb/s } \times 2 \text{ hops}}{11 \text{ Gb/s}} = 1.82 \text{ hops} \tag{3.2}$$

Communication performance depends on the network topology, the routing algorithm, and the match between application communication requirements and the network capacity. The issues of performance are further complicated in systems that support communication traffic priorities and QoS guarantees.

Deadlock and **livelock** are two conditions that drive network performance outside the expected bounds. A deadlock results from a cyclical resource dependency that prevents several traffic packets from making any further progress [48]. A deadlock occurs when packets form a cyclical dependency by simultaneously

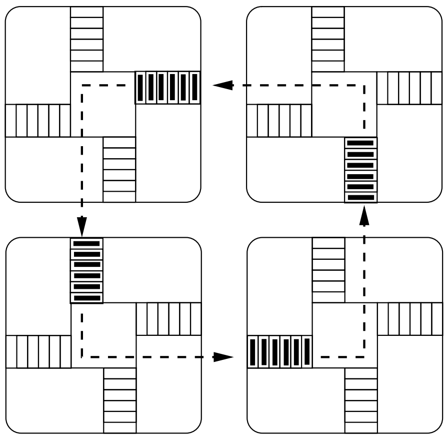


(a) Communicating task graph (b) Dim Order Routing (c) Adaptive Routing

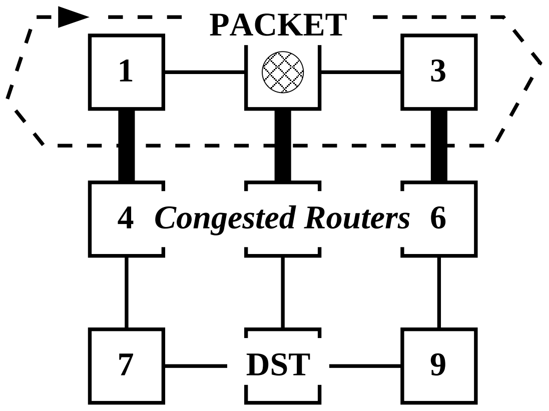
Figure 3.3: Routing algorithm determines average packet latency and saturation bandwidth of the network.

requesting the flit buffer space from a downstream router, but none can release the occupied space until the downstream packet does so (Figure 3.4(a)). This cyclical dependency cannot be resolved by the routers autonomously unless one of them gives up the request and drops its packet. Typically, a global deadlock detect and recovery mechanism is required to resolve the situation. If deadlocks are extremely rare, then "detect and recover" approach may be appropriate to allow for maximum routing adaptivity and flexibility. However, in a typical situation where the network is loaded close to its bandwidth saturation point, *i.e.* a number of in-flight packets in the network is large, it is best to employ deadlock *avoidance* schemes that include deadlock free routing.

A fully-adaptive routing algorithm that permits all possible turns in a packet's path can easily create a cyclical dependency between resources in adjacent routers.



(a) Deadlock: cyclical flit buffer space requests.



(b) Livelock: a router chooses the least congested path, and the packet does not make forward progress toward destination DST.

Figure 3.4: Deadlock and livelock impede forward communication progress.

As shown on Figure 6.2, prohibiting some turns to guarantee that a cycle can never be formed creates deadlock freedom. For a more detailed discussion of deadlock-free adaptive routing algorithm design, the reader is referred to [36, 46]. Many implementations use Virtual Channels or structured input flit buffers instead of prohibited turns. Every time a packet has to take a prohibited turn, it switches from its current virtual channel k to channel $k+1$ [58]. The virtual channels have the equivalent effect to prohibited turns, because they split monolithic physical resources into virtual resources and ensure that packets can never form a request dependency cycle with virtual resources.

A livelock, a special case of starvation, is a condition where a packet travels around the network, but makes no progress toward its destination [48]. Figure 3.4(b) illustrates the typical livelock. The packet in node 2 cannot get to its destination node DST because in the search for the path towards the destination, the distributed adaptive routing algorithm selects the least congested nodes 1 or 3. This continues and results in the packet circling around nodes 1, 2, and 3 as long as nodes 4, 5, and 6 remain congested. The packet may never reach its its destination. In addition to failing to deliver the packet, with livelock a packet consumes routing resources, increases congestion and degrades network performance. A *minimal* adaptive routing algorithms is the simplest approach to livelock-freedom. It only forwards a packet along topologically shortest paths and thus ensures that *every* router hop brings a packet closer to its destination. However, minimal algorithms have restricted adaptivity choices. For example, with a minimal router on a two dimensional mesh, a packet has at most two choices at every hop. A more complex approach is required to accommodate non-minimal (mis-routing) algorithms that guarantee livelock-freedom, such as switching from non-minimal to minimal routing as the packet ages.

The issues of connectivity, resilience, performance, deadlock and livelock can be addressed with several interrelated communication system features: topology, routing algorithm, flow control and buffer management. Topology and routing algorithms define the network properties and performance. Routing algorithm, buffer management, and switching dictate the NoC router architecture

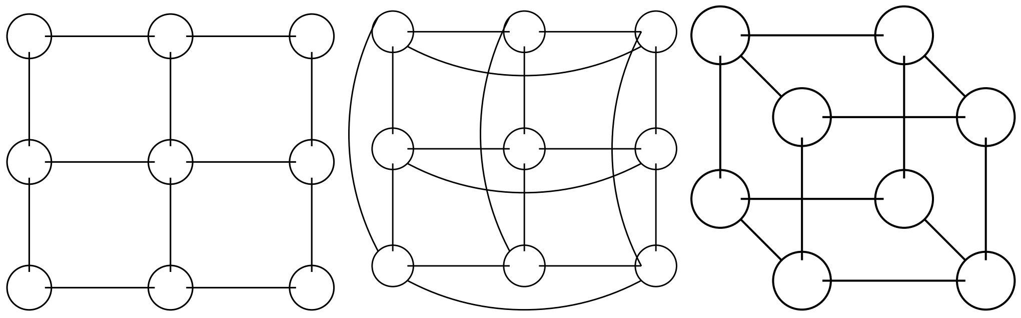


Figure 3.5: Examples of mesh and torus network topologies.

(Figure 3.1(b)). A router comprises input flit buffers, switch and routing/arbitration logic. Flit buffers receives packet flits[2] from the adjacent routers. The switch connects router inputs to the outputs directed by the routing/arbitration logic that makes decisions consistent with the routing algorithm and arbitration objectives (*e.g.* Round-robin or Maximum Weight Matching). The flit buffers play a critical role in the Voltage Frequency Island architecture by matching the rates between the adjacent routers and providing in-flight packet storage that optimizes communication channel and switch utilization. Typically as buffer sizes increase, fewer network resources are simultaneously committed to a single in-flight packet, when it is stalled waiting on an output channel [78]. Larger buffers reduce network congestion and improve its performance, but they also consume area and power, and their implementation requires a careful cost benefit analysis [47].

[2]Network packets are broken into *flits*, flow control units, for transmission.

Topology	Average Distance	Bisection B/W
Mesh	$\frac{2}{3}nk$	k^{n-1}
Torus	$\frac{1}{2}nk$	$2k^{n-1}$

Table 3.2: Performance bounds on (n, k)-mesh and torus

Topology is the key tool in the designer's arsenal to ensure that the network provides the required saturation bandwidth and average unloaded packet latency to satisfy application demands. The examples on Figure 3.3 illustrate the way the topology cross-section bandwidth and the routing algorithm together affect the network saturation bandwidth and average latency. The topology defines performance bounds on these critical metrics. Table 3.2 summarizes them for two popular, regular topologies — mesh and torus — shown on Figure 3.5. From the resiliency point of view, the network topology must provide sufficient *path diversity* to ensure that a routing algorithm can find a high capacity path around faulty or under-performing tiles. For a thorough discussion of different network topologies, we refer the reader to [34].

This work considers regular topologies for ease of implementation and testing of a two-dimensional die, comprising replicated compute/communication tiles. More exotic topologies, such as fat trees are complex to implement and provide a non-regular layout. We focus on a 2D mesh topology, which has a direct Manhattan layout for a collection of processing elements. Mesh routing algorithms are generally simple and require fewer virtual channels for deadlock-free adaptive routing. Meshes have peripheral nodes that, depending on the communication topology of the application, *may* result in uneven channel and router utilization. For example, a uniformly distributed communication traffic tends to create more congestion toward the center of the mesh than its edges.

The main difference between a mesh and a torus is additional "wrap-around" channels that eliminate the mesh edges. A two-dimensional torus would be an alternative topology alternative for a tiled multi-core architecture, but it is not studied in this work. [147] has shown that a torus can outperform a mesh on synthetic communication patterns due to its greater path diversity. However, the author did not consider additional challenges for VLSI layout or extra virtual channels required for deadlock free adaptive routing. For this work, the mesh is sufficient to demonstrate the way to translate the advantages of adaptive routing algorithms into smaller design margins that compensate for PVT variations. We argue that the mesh topology contains a subset of torus paths, and thus our mesh results represent the lower bound on the potential gains in a torus. At the same time, the additional implementation complexity of torus routing, an extra virtual channel for deadlock freedom, and a more complex VLSI layout, can negate the gains from adaptive routing.

A Routing Algorithm provides the mechanisms to deliver the performance potential of the topology by taking advantage of the network path diversity. Routing algorithms range in complexity and in the type and currency of the information that they use. In theory, with a resource unlimited implementation, a global, centralized routing algorithm can deliver the best performance: the lowest average packet latency and the highest saturation bandwidth. In practice, the networks are highly dynamic, and any attempt to globally schedule all communication flows in space and time for every packet transmission, switch and router setting step by step, is intractable. A heuristic approximation of a global schedule typically leads to results that are no better than significantly simpler distributed algorithms.

Distributed routing algorithms are organized in such a way that each router runs an independent instance of the algorithm and makes local decisions. We focus on these algorithms for two reasons: relative implementation simplicity and avoiding a single point of failure in the system. [34] offers a complete and detailed discussion of different routing algorithms. The following presents some key algorithm categories and their operating principles. The algorithms range from oblivious to fully-adaptive, which describes the degree to which an algorithm takes advantage of the path diversity in the topology. Another classification separates them in two categories: (1) the number of choices that an algorithm considers at each router step, and (2) outside information that is used to compute the next hop direction for a packet.

An oblivious algorithm considers a single output port for each packet based only on the packet's destination address. The algorithm does not react to changes in the surrounding traffic conditions. However, one can envision an implementation where an oblivious router attempts to balance the network load by randomly selecting the output channel along the packet's minimal path. This produces a livelock-free oblivious algorithm, but requires virtual channels for deadlock avoidance. In contrast, an adaptive algorithm considers a number of output paths at every NoC router and decides between possible paths based on various metrics. These may include any information obtained from the adjacent routers, such as traffic congestion [66], or local routing decision history. For deadlock freedom, an adaptive algorithm may limit its adaptivity with prohibiting certain packet turns or use virtual channels.

Naturally, oblivious algorithms are the simplest to implement and are the most scalable, since there is no sharing of information between the neighbor routers. A deterministic implementation such as Dimension Order router requires minimal hardware resources to achieve deadlock and livelock freedom, since all possible turns are known *a priori*, and the router algorithm can be proven to create no cyclical dependencies (Figure 6.2(a)). The adaptive algorithm implementation requires additional buffer and virtual channel management to avoid deadlocks, storage for router state and any information obtained from the neighbor routers, and, of course, a more complex algorithm logic. Chapter 7 discusses specific requirements for the algorithms explored in this work, and shows that even a relatively simple low overhead algorithm such as Minimal Adaptive Total Congestion (MATC) offers an impressive network performance improvement over the oblivious Dimension Order routing.

Buffer Management, Flow Control, and Switching implement deadlock avoidance and improve router switch utilization and network performance. The virtual channels complicate input flit buffer management and the inter-router flow control. A valid buffer management scheme separates the buffer space of each physical input into virtual channel partitions, such that no virtual channel can consume all memory resources and deadlock another VC. To ensure deadlock free operation under Virtual Cut Through (wormhole) switching, it is sufficient that every router input guarantees at *least one* flit buffer slot to each virtual channel.

Virtual channels also complicate the inter-router flow control. With a single channel, a simple handshake protocol (*e.g.* four phase [112]) would be sufficient for correct operation, albeit an inefficient choice if communication link is pipelined. A better option might be a credit based scheme that can maximize the channel transmission throughput [83]. However, with virtual channels, each VC requires

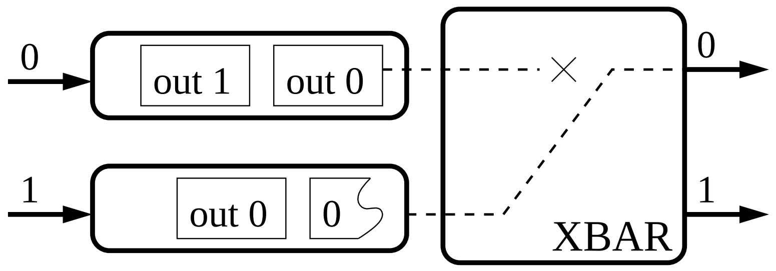


Figure 3.6: Head-of-Queue blocking.

an independent flow control to notify the sender of the downstream buffer space availability.

Input flit buffer management is also related to switching, which is the algorithm by which router input buffers are dynamically connected to the outputs. A switch is a passive router component controlled by the arbiter, which resolves the contention between multiple input channels vying for the same output, through a priority-based selection scheme. This “reduce” operation, which must fairly select a virtual input channel buffer from a set and usually involves an NP-hard matching problem, is complex to implement in hardware. An increase in the number of virtual channels usually corresponds to a significant increase in arbitration logic complexity [80].

Independent of the details of the arbitration scheme, all input queued routers suffer from Head-of-Queue blocking as a result of crossbar arbiter being able to only consider a single packet at the head of a FIFO. For example, consider Figure 3.6. The input buffer 0 contains two packets destined for output channels 0 and 1. Unfortunately, the output 0, which is requested by the packet at the head of the FIFO, is not available. The packet destined to output 1 is queued behind and cannot be accessed, although output 1 is available. The router switch is underutilized. Virtual Output Queues (VOQs) eliminate Head-of-Queue blocking, but complicate the arbiter implementation [93]. VOQs split the input buffers into a set of virtual buffers, one for each router output. An arbiter can simultaneously consider all output requests from all packets stored anywhere in the input buffers. VOQs preserve FIFO order between the packets destined for the same output channel. The arbiter implementation is more complex because it must now consider up n^2 possible connection requests, one for each virtual output.

Maximum Weighted Match (MWM) arbiter is guaranteed to deliver 100% switch bandwidth utilization although it is too complex for a practical hardware implementation [96]. Many heuristics that can be efficiently implemented in hardware achieve very comparable results to the MWM algorithm [45]. We have observed an improvement of up to 10% in network saturation bandwidth from routers with VOQ input buffers relative to those with simple FIFO input buffers. This improvement typically justifies the added implementation complexity, and that is the reason we used VOQ-enabled router implementation in this work (Section 5.2.1). The fundamental results that we obtained, *i.e.* the performance gains from the adaptive routing, would be qualitatively the same with or without VOQ input buffers.

3.3 Summary

Limitations in process scaling have forced the semiconductor industry to move toward parallel multi-core architectures. With respect to fault and PVT variation resilience, these architectures provide huge opportunities with core redundancy and regularity. A globally asynchronous NoC design connecting independent voltage and clock core/router tile domains, combined with architecture controls enable the run-time task mapping and routing to deliver the average rather than the worst case performance.

This chapter discussed core/router tile micro-architecture and identified its salient features that enable introspective run-time mapping and allow the run-time system to navigate through tile's PVT induced operating range. These features may include simple distributed run-time indicators such as fault detectors and checkers, tile operating range characterization circuits, performance counters, and flow control. The chapter focused the critical features of a Network-on-Chip architecture, including: connectivity and resilience, performance, deadlock and livelock freedom. Further, it demonstrated the way designers can address these concerns through careful selection of network topology, routing algorithm and buffer management strategies. These related network properties determine the NoC implementation complexity, performance, and the quality of service guarantees that the network delivers to the run-time system.

Chapter 4

Sparing: Redundancy and Multicore

The architecture discussed in Chapter 3 isolates cores from one another via distinct clock, voltage domains, and an asynchronous network. Each core runs at its own rate and contains faults and performance variations within. This architecture results in an inherently redundant fault and performance variation isolation model that can dramatically improve manufacturing functional and parametric yield. After manufacturing, during power up or dynamically, the system can test and disable faulty and under-performing cores, and select a *subset* of cores to run an application. To ensure that a die contains the required number of functional cores, spare cores can be designed into a multi-core architecture.

This is not a novel idea. Fine-grain sparing of rows and columns is used in DRAM [130]. Currently, coarse-grain sparing is employed in Cisco Metro that has 4 redundant processor cores per die; the STI Cell processor has 8 cores, but only 6 are enabled in its mainstream consumer application in the Sony Playstation 3 [51, 110]. Die level sparing was also investigated with block redundancy for wafer scale integration [63].

The question now becomes one of how to best utilize chip resources and introduce redundancy to improve chip yield. We develop a model for a multi-core die yield and cost with core sparing, and examine several questions:

- What are the bounds of core sparing?
- How does core redundancy compare to module redundancy and semiconductor device over-engineering?
- With the trends in semiconductor technology, what is the *optimal* core area that minimizes the number of faulty on-chip cores and maximizes the yield?
- What is the relationship between the the area overhead of multi-core architectures and the yield?

4.1 Resource Redundancy

As the semiconductor device size shrinks, system yield and reliability become increasingly difficult to manage due to a combination of defects and parametric variations. Although process and manufacturing engineers are constantly finding new ways to optimize yield through improved accuracy and precision of semiconductor processing steps, the quantities of substances used approach tens of atoms, and are impossible to control reliably. When dealing with a stochastic processes and such small sample size, redundancy must be used effectively to manage performance and yield. We examine two critical questions: (1) where to use redundancy and (2) how to manage it? The following classifies different schemes and analyzes them based on cost, implementation, and the impact on PVT variations and yield.

Device Over-engineering and DFM rules. To improve yield and resilience to process variation, engineers employ device and interconnect widths and inter-device spacings larger than required by the design rules. Although this technique has a significant permanent impact on area and performance, it reduces variance on such parameters as transistor threshold voltage and line edge roughness [22]. Increased feature sizes reduce the critical area affected by defects and thus can significantly improve the yield, assuming the equivalent defect density and size distribution. This is consistent with ITRS predictions that show that defect density has saturated and remained constant as the semiconductor process nodes shrink in size [1].

This technique must be used sparingly due to its high cost. Depending on implementation details and specific DFM rules, the impact on the area can vary from linear to quadratic. The technique is also accompanied by a linear increase in delay and power consumption with respect to the design margin. Due to its static nature, decisions about chip area allocation must be made at design time and for the worst case. This renders device over-engineering less effective in future semiconductor generations when tiny physical dimensions preclude our ability to carefully control the manufacturing process and force the safety margins to grow.

Fine grain circuit redundancy With greater uncertainty in the manufacturing process, circuit techniques such as error-correcting codes (ECC) or double module redundancy (DMR) can detect and correct both manufacturing and transient faults. With redundant circuits, the designer can contain the problem to avoid its propagation through the system. With appropriate coding the area overhead of these techniques can be smaller than that of device over-engineering. Still, they incur a permanent cost on circuit performance, even if they do not lengthen the critical path. Redundant circuits increases the number of near critical paths, which due to process variations increase the probability of a long outlier path that determines the system performance. At the circuit level, the combination of overclocking and error correction code to compensate for variations and defects can sometimes optimize overall power/performance and yield [13]. As with device over-engineering, decisions about where to add redundancy or which codes to use must be made early at design time and for the worst case.

Module redundancy There are two principle differences between fine grain circuit redundancy and module redundancy. First, in larger modules, one typically encounters resource replication (*e.g.* DMR) rather than code-based error correction because it is a more natural and general solution. Coding works well on small arithmetic and communication circuits, but it is non-trivial for complex modules. Second, the larger module size allows for more flexible run-time resource management controlled in hardware or even in software.

For example, consider the following fault management scenarios on a super-scalar processor with several redundant ALUs. Scenario 1: every instruction runs through three distinct ALUs and the majority result is used [131]. Scenario 2: only some instructions (perhaps in the "reliable" section of a program) run in this TMR mode. Scenario 3: temporal redundancy (running instructions several times through the same ALU) used on some or all instructions [8]. Scenario 4: normal mode, all instructions run once through an ALU to maximize instruction throughput and processor performance. Hardware or software can select a scenario at run time and flexibly utilize these redundant ALUs for fault-resilience or raw computation.

Device over-engineering, fine grain circuit and module redundancy share two common problems: (1) interdependence and (2) spatial correlation. First, since the redundant devices, circuits and modules form a part of a larger design, their faults and performance variations can propagate through the entire system. Second, since redundant components (*e.g.* ALUs) are typically located in the same part of the die, they are identically influenced by spatially correlated performance variations and defect clustering. For example, *all* redundant components might be under-performing according to a specification, rendering the entire die non-functional. These problems combine to reduce die performance and yield. An ideal solution would maximally isolate the redundant components from each other and distribute them across the chip. Although this does not necessarily imply a multi-core architecture, multiple cores per die is the most natural and manageable solution.

Core-level redundancy We focus on processor cores, although different cores would not qualitatively affect our discussion. Globally Asynchronous Locally Synchronous (GALS) implementation results in both functional and performance isolation between the cores. With redundant cores, the system can be fault-tolerant. Two aforementioned problems are solved. First, there is no core interdependence, the faults and variations within a die area occupied by a core only affect it but not its peers. Second, the cores are distributed all over the die and operate independently at *different* performance points dictated by the intra-die variations. In fact, the performance spectrum of cores covers the spectrum of on-die PVT variations.

After manufacturing the chip is tested, and faulty cores can be disabled with fuses. An inter-core communication protocol must ensure that disabled cores

Parameter	Description (default/typical value)
C_{die}	die cost
C_{wafer}	processing cost for a wafer (\$1250[1])
Y_{die}	die yield
Y_0	gross wafer yield (1)
Y_{core}	core yield (a function of core area)
A_{die}	die area
D_{wafer}	wafer diameter (300mm)
D_0	avg defect density (0.0002/mm^2)
α	defect clustering factor (3)
A_o	*per core* infrastructure overhead ($1mm^2$)
A_f	total fault-free, functional die area
N_f	total number of fault-free cores

Table 4.1: Parameters used in the models. Default values shown in parenthesis are based on ITRS 2005 [1] and used to obtain the experimental results.

simply appear unavailable and are not used by their neighbors. Unlike module redundancy, an OS can manage a multi-core essentially with existing scheduling techniques. For example, a task scheduling algorithm (*e.g.* task stealing) can automatically adopt to a system with cores with heterogeneous performance, since each core processes tasks at its own rate. The defective cores do not run at all, and since they do not remove tasks off a common queue, they appear busy or unavailable. The core performance, processing throughput, is exposed to the run-time environment as flow control as discussed in Section 3.1.

[1]Although the wafer cost shown maybe out-of-date, its actual value does not affect the qualitative results and conclusions of our analysis.

4.2 Traditional Chip Cost Model

Th following is a short review of the traditional defect-free die cost model, based on the yield model with negative binomial defect distribution [108, 111]. Table 4.1 summarizes all model parameters in this chapter. The die yield can be described as the probability that n defects hit the critical area A given an average defect distribution D_0:

$$p(n, A, D_0) = \frac{\Gamma(\alpha + n)}{n!\Gamma(\alpha)} \frac{(AD_0/\alpha)^n}{(1 + AD_0/\alpha)^{n+\alpha}} \tag{4.1}$$

We are interested in *defect-free* area, $n = 0$:

$$p(0, A, D_0) = (1 + AD_0/\alpha)^{-\alpha} \tag{4.2}$$

Our die cost model ignores test and packaging costs because they are not relevant to the discussion:

$$C_{die} = \frac{C_{wafer}}{Dies/Wafer \times Y_{die}} \tag{4.3}$$

The wafer cost is fixed and outside of our control, but the die area determines both $Dies/Wafer$ and die yield Y_{die}:

$$Dies/Wafer = \frac{\pi \times D_{wafer}}{\sqrt{2A_{die}}} \tag{4.4}$$

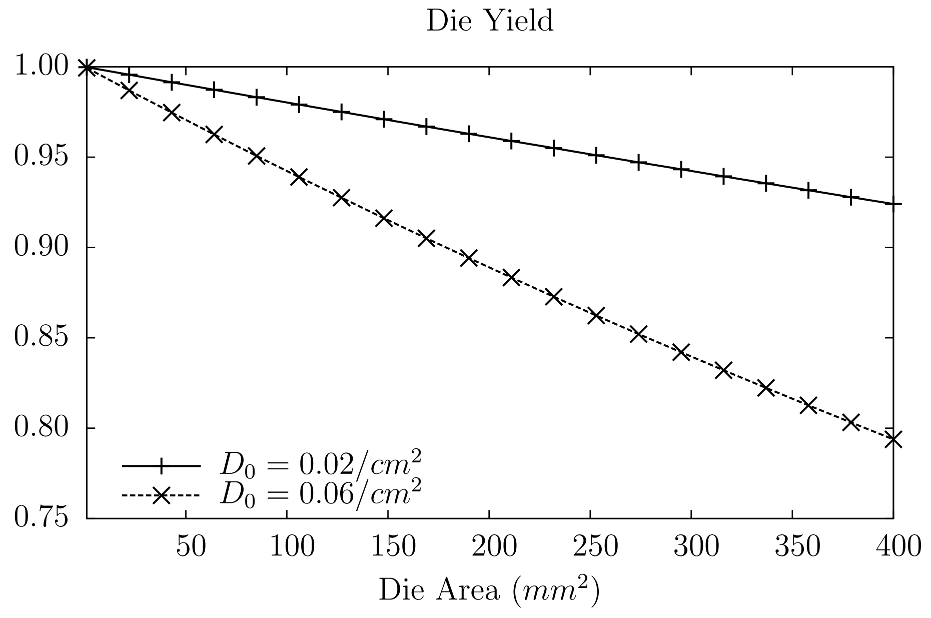
$$Y_{die} = Y_0 \times p(0, A_{die}, D_0) \tag{4.5}$$

This assumes that the the entire die A_{die} is the critical area, which gives a pessimistic yield prediction but does not affect the trends discussed in the work. Figure 4.1(a) illustrates the expected inversely correlated relationship between the die yield Y_{die} and the die area A_{die} for sample parameter values from ITRS 2005 [1]. Intuitively, as the die area increases, the probability of one or more defects occupying that area grows and reduces the die yield (Equation 4.2). We combine A_{die} and C_{die} and obtain the die cost:

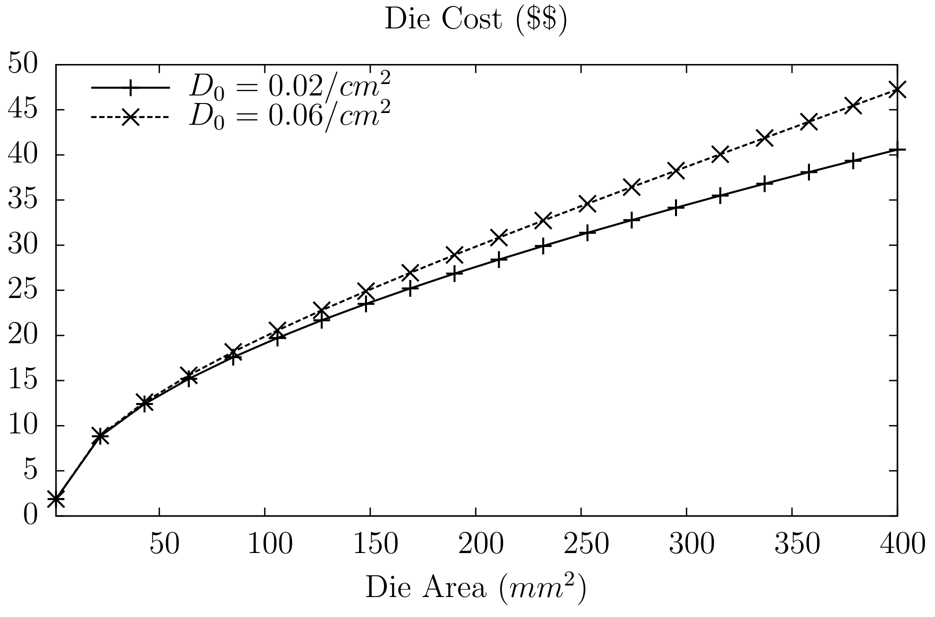
$$C_{die} = \frac{C_{wafer}\sqrt{2A_{die}}\left(1 + \frac{D_0 \times A_{die}}{\alpha}\right)^{\alpha}}{\pi \times D_{wafer} \times Y_0} \tag{4.6}$$

$$C_{die} \sim O\left(A_{die}^{\alpha+\frac{1}{2}}\right) \tag{4.7}$$

Figure 4.1(b) illustrates the relationship between the die cost and area. The cost grows as a high-degree polynomial function of the die area. With a typical value of the defect clustering factor $\alpha = 3$, $C_{die} = f(A_{die}^{3.5})$.



(a) Die Yield



(b) Die Cost

Figure 4.1: The relationship between die area and its yield and cost.

It is important to note that Equation 4.5 in practice represents the yield of a single die layer, *e.g.* a metal layer or a mask, as it is impacted by defects and impurities during its semiconductor processing step. This expression is sufficient for our analysis, since we strive to demonstrate that the reduction in core size significantly improves core yield and thus reduces the overall die cost. Accounting for multiple semiconductor processing layers, the die yield is a product of individual layer yields:

$$Y_{multilayer\_die} = Y_0 \times \prod_{i \in DieLayers} p(0, A_i^c, D_0) \tag{4.8}$$

where Y_0 is gross wafer yield, and A_i^c is the *critical* area for layer i. The critical area is defined as the area of the die (typically, a fraction of total die area), where a defect can produce a fault, such as a short. Section 4.4 presents a more detailed discussion on the critical area, but intuitively, on the same die the critical area of an upper metal layer (*e.g.* M9) is smaller than that of a bottom metal layer (*e.g.* M1) for the same defect size distribution.

For our analysis of multi-core sparing, we continue to use Equation 4.5 for die yield as it is simpler and die independent, but accurately captures the relevant trend.

4.3 Sparing Defective Cores

Architectures with redundant cores can disable faulty cores and operate a subset of a die. The area overhead of such an architecture stems from independent core-level power delivery and clock networks, and inter-core communication infrastructure that includes synchronization, arbitration for shared resources (*e.g.* buses, links), *etc.* Let us designate A_o to be *per core* overhead in this architecture. If a designer requires total functional, fault-free die area A_f, how should the chip be partitioned into cores to minimize the die cost? If N_f is the number of functional equally sized cores, and each core incurs a fixed area overhead A_o, then:

$$A_{core} = \frac{A_f}{N_f} + A_o \tag{4.9}$$

$$A_{die} = A_{core} \times \frac{N_f}{Y_{core}} = A_{core} \times \frac{N_f}{p(0, A_{core}, D_0)} \tag{4.10}$$

The total die area A_{die}, which determines the cost, comprises $N = \frac{N_f}{Y_{core}}$ cores: N_f functional and $(N - N_f)$ defective ones. When defective cores are disabled, the die contains N_f defect-free cores with the aggregate area A_f. Substituting Equation 4.5 for core yield into the expression for the total die area, we obtain:

$$A_{die} = N_f \left(\frac{A_f}{N_f} + A_o\right)\left(1 + \frac{D_0}{\alpha}\left(\frac{A_f}{N_f} + A_o\right)\right)^{\alpha}$$

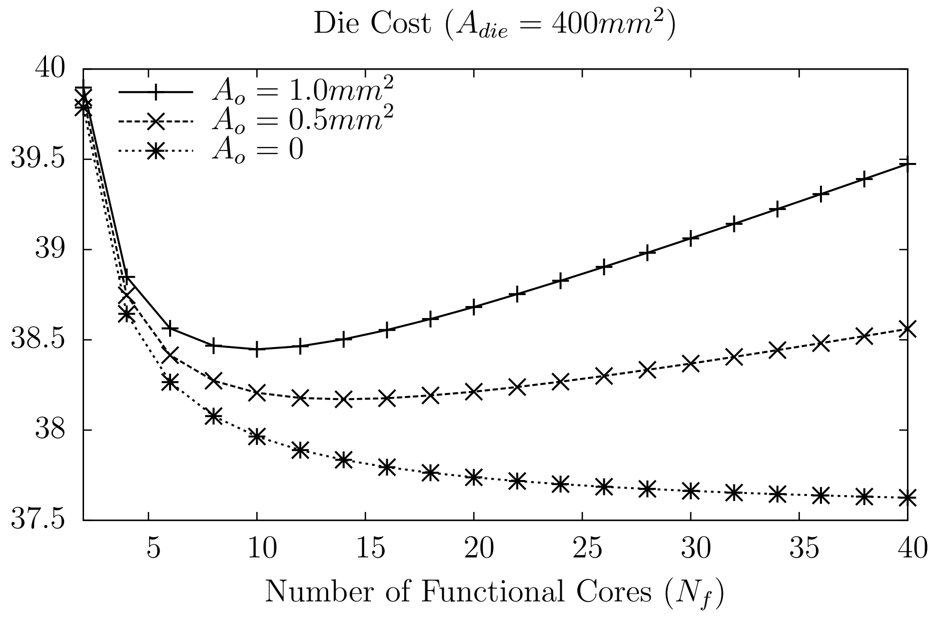


Figure 4.2: Die cost *vs* the total core count for different overheads.

Each manufactured N-core die contains the desired N_f fault-free cores, and therefore $Y_{die} = 1$. The trade off is that core sparing scheme produces larger but perfect yielding die, but fewer of them per wafer. Let us designate Y_0 as gross yield to capture external factors such as wafer and facility yield. The total die cost is constant for a given total die area:

$$\begin{aligned} C_{die} &= \frac{C_{wafer}}{Dies/Wafer \times Y_0} = \frac{C_{wafer} \times \sqrt{2A_{die}}}{\pi \times D_{wafer} \times Y_0} \qquad (4.11) \\ &= \frac{C_{wafer}\sqrt{2N_f\left(\frac{A_f}{N_f} + A_o\right)\left(1 + \frac{D_0}{\alpha}\left(\frac{A_f}{N_f} + A_o\right)\right)^{\alpha}}}{\pi \times D_{wafer} \times Y_0} \end{aligned}$$

Figure 4.2 illustrates the equation above for three example values of per core area overhead A_o. With a few but large functional cores, the die cost is high due to low core yield pushing the total die area up. In an ideal system without any overhead ($A_o = 0$), the die cost decreases as the number of cores per die grows because the cores become smaller and yield well. But considering the overhead of Voltage Frequency Islands and the communication fabric, as the number of cores grows, the overhead of many small cores dominates the die area. For each overhead value of $A_o \neq 0$, a unique N_f minimizes the total die area and thus the die cost by balancing the competing improvements in core yield with the increased overhead. We solve $\frac{\partial A_{die}}{\partial N_f} = 0$ to determine the optimal $N_{f,opt}$:

$$N_{f,opt} = A_f \times \frac{D_0 \left(\frac{\alpha-1}{\alpha}\right) + \sqrt{D_0^2 \left(\frac{\alpha+1}{\alpha}\right)^2 + \frac{4D_0}{A_o}}}{2\left(1 + \frac{D_0 \times A_o}{\alpha}\right)}$$

The optimal core size is then:

$$A_{opt} = \frac{A_f}{N_{f,opt}} = \frac{2\left(1 + \frac{D_0 \times A_o}{\alpha}\right)}{D_0 \left(\frac{\alpha-1}{\alpha}\right) + \sqrt{D_0^2 \left(\frac{\alpha+1}{\alpha}\right)^2 + \frac{4D_0}{A_o}}}$$

The optimal core size that minimizes total die area — both functional and faulty cores — depends only on the defect density D_0, semiconductor process defect clustering parameter α, and multi-core architecture implementation overhead A_o.

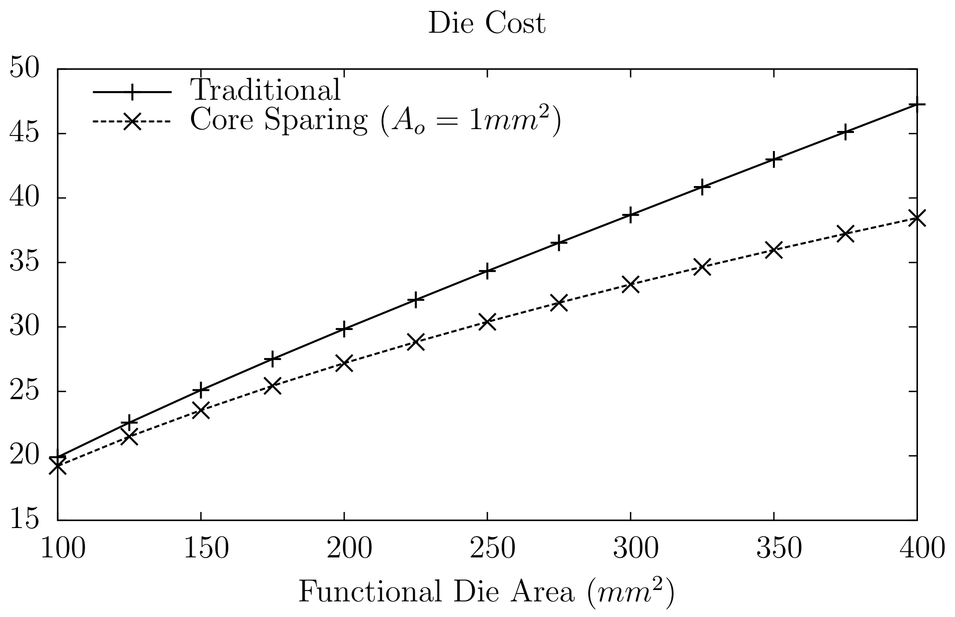


Figure 4.3: Die cost for the traditional *vs* core sparing dice.

Substitute $N_{f,opt} = \frac{A_f}{A_{opt}}$ into Equation 4.11 to obtain the total die cost with optimally sized cores:

$$C_{die} = \frac{C_{wafer}}{\pi D_{wafer} Y_0} \times \sqrt{\frac{2A_f}{A_{opt}} (A_{opt} + A_o) \left(1 + \frac{D_0}{\alpha}(A_{opt} + A_o)\right)^{\alpha}} \quad (4.12)$$

$$C_{die} \sim O\left(\sqrt{A_f}\right) \quad (4.13)$$

Multi-core architectures that enable core sparing asymptotically reduce chip cost and provide flexibility for yield management at manufacturing, deployment and run time. Figure 4.3 illustrates the relationship between die cost C_{die} and the functional area A_f for the traditional and core sparing architectures, where

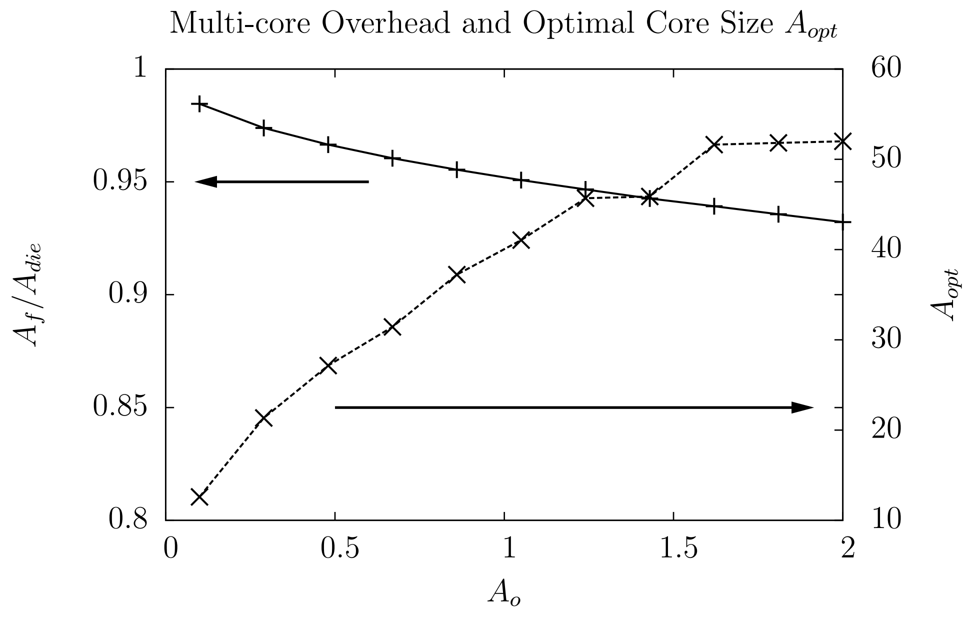


Figure 4.4: The ratio of functional A_f to the total die area A_{die} is sensitive to multi-core overhead A_o.

the latter uses the cores of the optimal size A_{opt}. One can observe the clear difference in the asymptotic behavior. Figure 4.4 illustrates the overhead of multi-core architecture with sparing as the ratio $\frac{A_f}{A_{die}}$ of functional area to the total die area, which comprises defective cores, $A_o \times N$ area overhead and the functional die area. The figure also shows that the optimal core size A_{opt} increases with the overhead A_o, as expected to balance the cost of the core isolation and decoupling.

4.4 Yield and Over-engineering

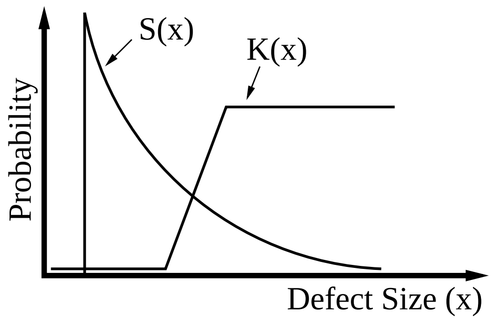
Core sparing results in an asymptotic cost improvement, but how does it compare with redundancy from over-engineered semiconductor devices? An accurate general yield model that relates the transistor and interconnect sizes and the chip yield is difficult to create without circuit structure and layout details. The model below captures the trends sufficient for our analysis. The yield depends on the defect-sensitive critical die area A_C [23]:

$$A_C = A_{die} \int_0^\infty K(x)S(x)dx$$

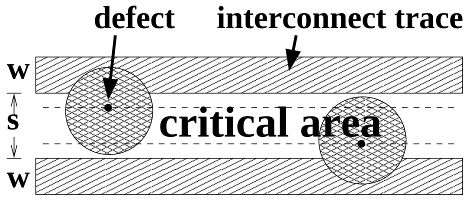
where x is defect diameter, $K(x)$ is the fault probability kernel, and $S(x)$ is defect size distribution. Assume that defects have a circular shape. $K(x)$ and $S(x)$ depend on the process feature size as illustrated in Figure 4.5(a). The actual values of these parameters are not critical to our discussion, but the overall monotonicity is. Defect size distribution $S(x)$ can be defined as [54]:

$$S(x) = \begin{cases} 0, & \text{if } x < x_0 \\ \frac{k}{x^3}, & \text{otherwise} \end{cases}$$

where x_0 is the minimum defect size, *i.e.* the defects smaller than x_0 do not result in faults and have no impact on yield. Fault probability kernel usually has the



(a) Functions $S(x)$ and $K(X)$.



(b) The critical area for a "short" fault is between the dashed lines.

Figure 4.5: Critical area parameters

following definition:

$$K(x) = \begin{cases} 0, & \text{if } x < \lambda_0 \\ f(x), & \text{if } \lambda_0 \leq x < w_{max} \\ 1, & \text{otherwise} \end{cases} \tag{4.14}$$

where $0 \leq f(x) \leq 1$ is a monotonically increasing function with respect to x that represents the probability that a defect of size x creates a fault. A defect larger than the maximum size w_{max} will result in a fault independent of its location on a die. Figure 4.5(b) illustrates a circuit "short" fault in the interconnect structure, where all interconnect segments are parallel to one another. For this fault $f(x) = x - s$ and $w_{max} = w + s$.

Consider die area A_{die} with feature size of 2λ. A crude estimate of the total device count (TDC) that basically includes transistors and interconnect traces on the die is:

$$TDC = \frac{A_{die}}{s_0 \lambda^2}$$

where s_0 is the actual device size in terms of λ^2 squares (*e.g.* a minimum size transistor could be 6–8 λ^2). When a designer increases device feature sizes by a factor of F to improve resilience to faults and variations, the total device count drops quadratically for the same die area while the expected operating frequency decreases linearly [112]:

$$\begin{aligned} TDC &= \frac{A_{die}}{s_0 (F\lambda)^2} \\ Freq &= \frac{f_0}{F} \end{aligned}$$

How does F affect the yield? As F increases, defect size distribution $S(x)$ does not change for a given process. However, the fault probability $K(x)$ decreases because λ_0 is replaced with $F\lambda_0$ in Equation 4.14. Since $f(x)$ is monotonically increasing, we can approximate $K_F(x)$ with unit step function to get the worst case fault probability:

$$K_F(x) = u(x - F\lambda_0) = \begin{cases} 0, & \text{if } x < F\lambda_0 \\ 1, & \text{otherwise} \end{cases} \tag{4.15}$$

The critical area A_C affected by defects:

$$\begin{aligned} A_C &= A_{die}\int_0^\infty K_F(x)S(x)dx &(4.16)\\ &= A_{die}\int_0^\infty u(x-F\lambda_0)S(x)dx &(4.17)\\ &= A_{die}\int_{F\lambda_0}^\infty \frac{k}{x^3}dx &(4.18)\\ &= A_{die}\frac{k}{2F^2\lambda_0^2} &(4.19)\end{aligned}$$

For defect size distribution $S(x)$ to be a valid probability density function, it requires that $\int_{-\infty}^{\infty} S(x)dx = 1$ and thus $k = 2x_0^2$. Assume, for example, that minimum defect size $x_0 = \frac{1}{2}\lambda_0$, to obtain:

$$A_C = A_{die}\frac{2\frac{1}{4}\lambda_0^2}{2F^2\lambda_0^2} = A_{die}\frac{1}{4F^2} \qquad (4.20)$$

which suffices to illustrate our point that the critical area decreases as device features grow by a factor of F.

The reduced critical area, however, comes at the expense of a lower on-die device count and lower clock frequency. To make a fair metric, instead of focusing on the die cost, we maximize the product of the total device count (TDC) and clock frequency. In effect, this maximizes the number of functional devices and their speed in a fixed die area, *i.e.* the total chip functionality or features. Let N

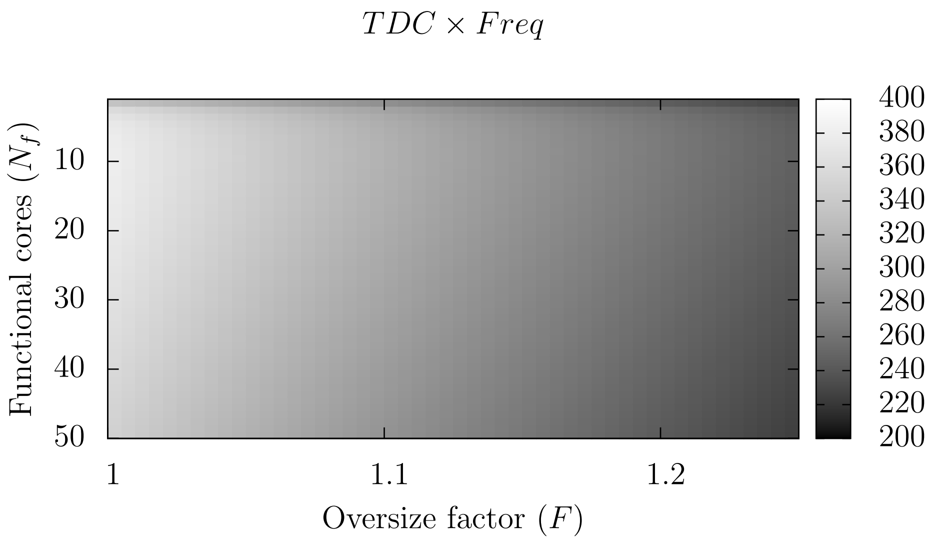


Figure 4.6: Device oversizing *vs* the number of functional cores on a $A_{die} = 400mm^2$. The values of $N = 11$, $F = 1$ maximize $TDC \times Freq$.

be the total number of on-chip cores, both faulty and functional. Setting $s_0 = 1$ and $f_0 = 1$ for simplicity (they do not affect the qualitative result), we obtain:

$$A_{core} = \frac{A_{die}}{N} - A_o \tag{4.21}$$

$$Y_{core} = Y_0 p \times \left(0, \frac{A_{core}}{4F^2}, D_0\right) \tag{4.22}$$

$$TDC = N \times Y_{core} \times \frac{A_{core}}{4F^2} \tag{4.23}$$

$$Freq = \frac{1}{F} \tag{4.24}$$

$$\text{maximize} \quad TDC \times Freq \tag{4.25}$$

Figure 4.6 illustrates $TDC \times Freq$ for $A_{die} = 400mm^2$. The graph shows that to maximize the number of fault-free transistors running at the highest possible clock frequency, the parameter values should be $F = 1$ and $N = 11$, which assumes the process $D_0 = 0.0002/mm^2$, $\alpha = 3$ and the overhead $A_o = 1mm^2$. This demonstrates over-engineering to be ineffective to improve die yield. There is no closed form expression for F that demonstrate that $F = 1.0$ (no device oversizing) always maximizes the on-die device count and frequency product, it is the case on a range of parameters we analyzed. Although over-engineering can improve the yield for a fixed die area, the reduction in the device count and the frequency undermines the gains. Core spares are more effective to maximize chip functionality.

4.5 Circuit and Module Redundancy

Although redundant circuits and modules do not offer the same resource management flexibility to the OS as the redundant cores, there are situations where redundant circuits are very effective (*e.g.* DRAM columns).

Our yield model for cores with redundant resources (*e.g.* ALUs) depends on two parameters: $r \in [0, 1]$ – the fraction of the area fortified by redundancy, $R \in [1, \infty)$ – area replication factor that describes the overhead. For example, if 20%

of the area is consumed by ALUs, implemented with Dual Modular Redundancy (DMR), then $r = 0.2$ and $R \approx 2$, which ignores the overhead of the comparator.

If a designer requests the fault-free area A, then the total core area including the redundant modules is:

$$A_{total} = A(1-r) + ArR = A(1-r+rR) \tag{4.26}$$

Area $A(1-r)$ is not affected by redundancy, but area ArR is reinforced. The yield for die area A_{total} with the redundant modules tolerant of up to N faults, is:

$$Y \;=\; p\left(0, A\left(1-r\right), D_0\right) \times \sum_{n=0}^{N} p\left(n, ArR, D_0\right) \tag{4.27}$$

Combining with the multi-core yield model, we obtain the total die area for a system with core sparing and within core module redundancy. If the design requires total functional area A_f that is partitioned into N_f cores, then

$$\begin{aligned} A_{core} &= \frac{A_f}{N_f}\left(1-r+rR\right) + A_o \\ Y_{core} &= p(0, A_{core}(1-r), D_0) \times \sum_{n=0}^{N} p(n, A_{core}rR, D_0) \\ A_{die} &= \frac{N_f}{Y_{core}} \times A_{core} \end{aligned}$$

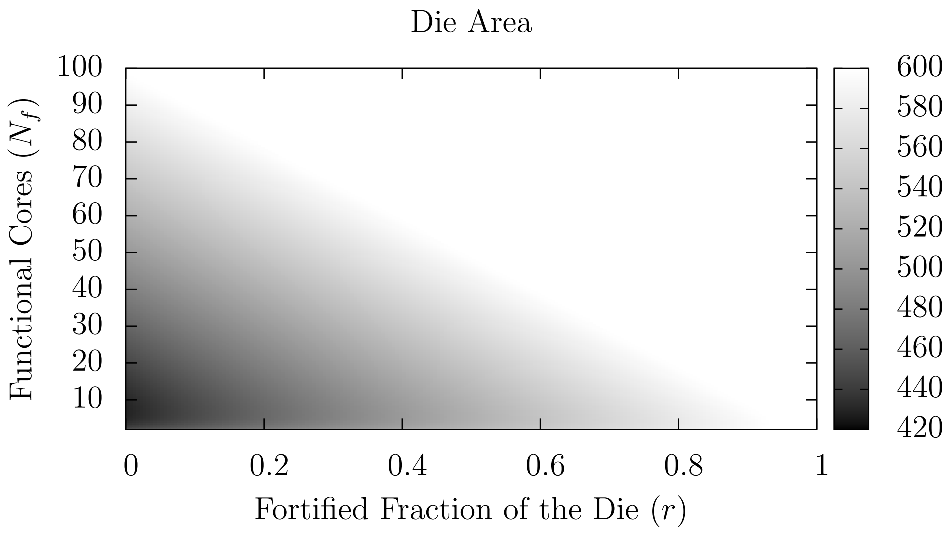


Figure 4.7: The total die area *vs* r and N_f. $(A_f = 400, A_o = 1, R = 1.5)$

Figure 4.7 illustrates the relationship between the total die area and the choices for r and N_f. The total die area includes the fault-free cores and the spared faulty cores. Each functional core provides A_f/N_f of fault-free area, but occupies larger area A_{core} due to redundant modules (ArR) and multi-core overhead (A_o).

Figure 4.8 shows that the dominant factors — redundancy overhead R and defect density D_0 — that determine the optimal combination of r and N_f to minimize total die area A_{die}. There are two regions shown in the graphs.

1. When the redundancy overhead R is very low, it suggests that the modular redundancy should be used on the entire core $(r = 1)$. The modular re-

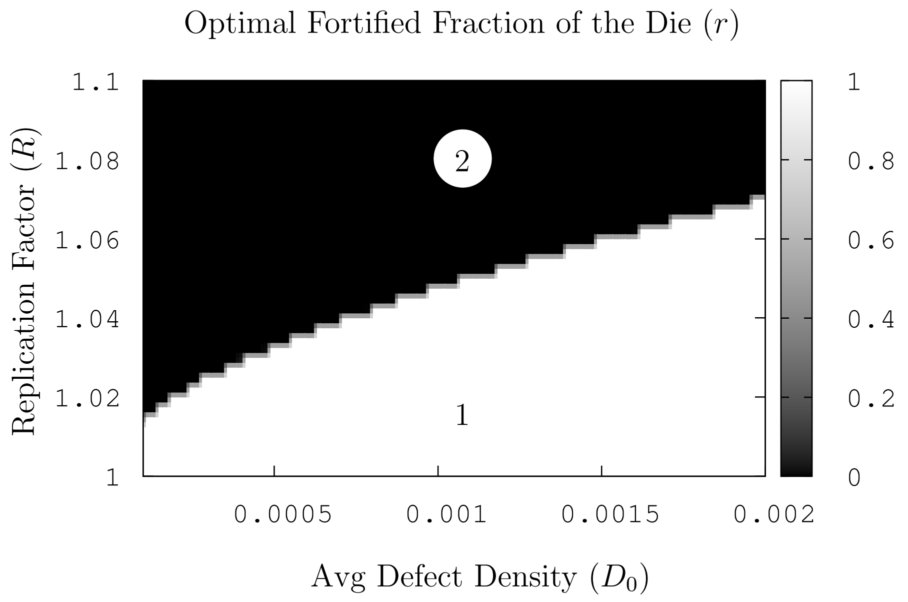
dundancy is a more effective to improve the die yield as compared to core sparing, as evidenced by the low value of N_f.

2. In contrast, if the overhead of modular redundancy is high (exceeding 2–4%), core sparing is a superior solution.

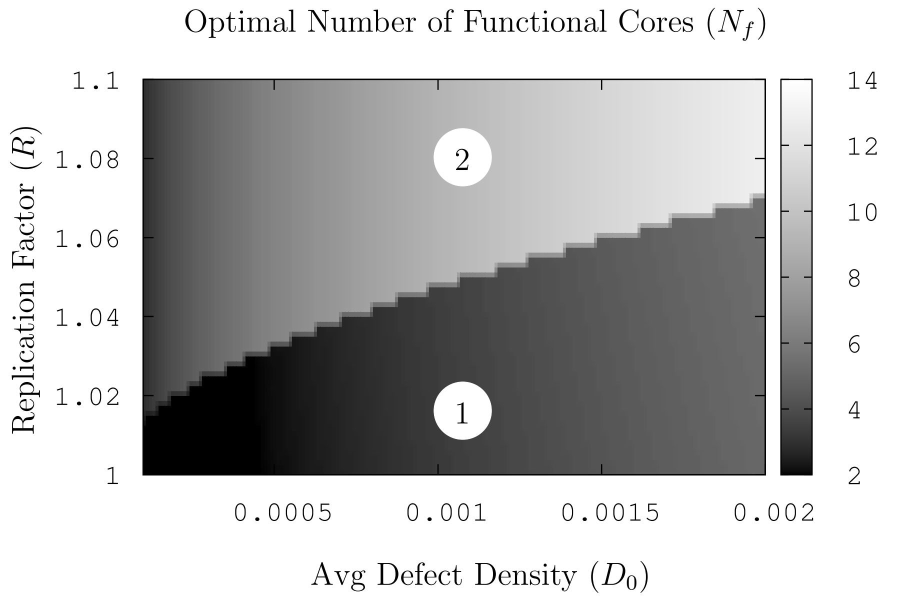
Notice that the boundary between two regions changes as expected when average defect density increases. The analysis decisions must be made based on the intrinsic process yield parameters.

4.6 Model Comparison

Figure 4.9 compares the die cost for different redundancy levels: semiconductor device over-engineering, redundant modules within a core, and core sparing. Each curve represents the die cost assuming the optimal parameters: core size A_{opt}, reinforced fraction of the core area r, and over-engineering factor F. The x-axis shows the functional, defect-free area as would be requested by a designer. Core sparing consistently results in the minimum die cost compared to other schemes. Based on our models, the best use of the die area is as set of independent cores, some of which can be left as spares. Other forms of redundancy do not provide sufficient yield boost to compensate for their area overhead.



(a) Optimal r *vs* replication factor R and average defect density D_0.



(b) Optimal N_f *vs* replication factor R and average defect density D_0.

Figure 4.8: Optimal (r, N_f) parameters to minimize the total die area A_{die} for functional area $A_f = 400$.

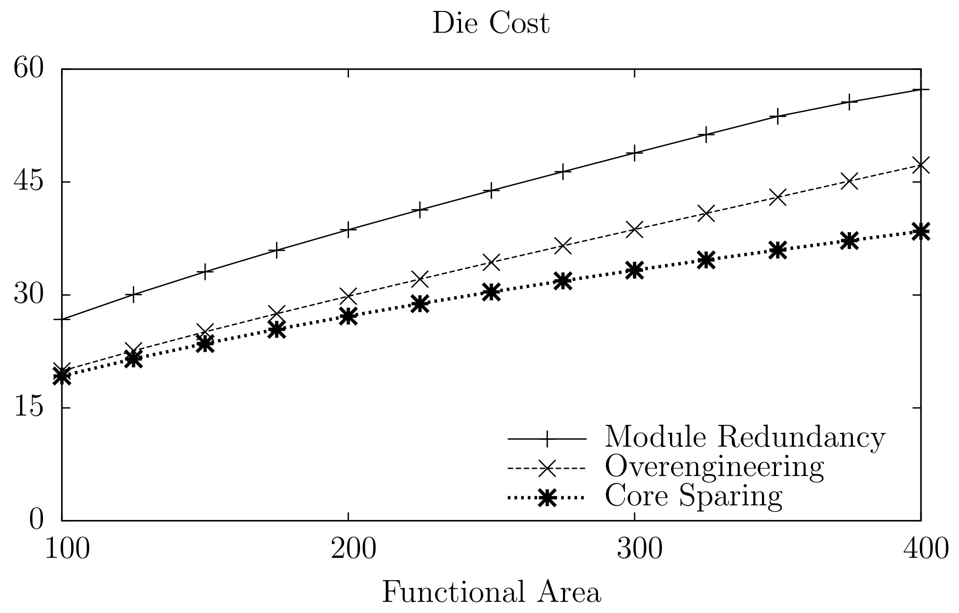


Figure 4.9: Die cost comparison with different schemes. Parameters: $A_o = 1\text{mm}^2$, $\alpha = 3$, $R = 2$.

Section 4.3 demonstrates that core sparing results in $C_{die} \sim O(\sqrt{A})$ if the optimal core area is used. The optimal core area depends only on the process parameters: defect density (D_0), process complexity α, performance variations (σ), and the area overhead (A_o). A_o is the result of independent clock and voltage domains, and the asynchronous communication fabric. Without this overhead, simply selecting the smallest possible core size would minimize the die cost, since the probability of a defective or slow core decreases with its size.

Even ignoring the overhead, one cannot select an arbitrarily small core, since we require a core with a minimum functionality and compute power. Modern CAD tools and compilers cannot partition or parallelize applications into arbitrary sized communicating components. However, these tools cannot be blamed entirely, since many classes of applications do not partition arbitrarily, but naturally divide into coarse-grain tasks or very fine-grain logical functions, but nothing in between. This limitation places application specific restrictions on useful processor core sizes, which in turns affects the attainable die yield with core sparing technique.

What is the typical processor core area today? Existing commercial and academic multi-core architectures show that a super-scalar processor core ranges from 27–51$G\lambda^2$, while a simple, in-order core requires 0.5–11$G\lambda^2$ of silicon area [56, 135, 142, 44, 115, 102]. The rest of the die typically contains caches, which are protected by ECC. Architects striving to optimize the die yield have a range of options for core areas. They can select a core close to the optimal A_{opt} that meets their application requirements and also minimizes the impact of defects and performance variations.

4.7 Summary

Redundancy in some form is and will increasingly be necessary to combat manufacturing defects and performance variations. The presented models help to answer several critical questions. Where and how to use redundancy in circuit and architecture design? Should we strive toward fewer reliable cores with redundant modules and circuits, or make a greater number of smaller but possibly faulty and unreliable cores? The core reliability in the first option incurs the performance, power and area overhead as compared to the alternative, but as we have shown it does not reduce the die cost.

Our simple model of device over-engineering and module redundancy suggests the latter: many unreliable, smaller high-performance cores. However, to answer this question completely, one must investigate the relationship between delay, area, power, error correction and compensation techniques enabled by each type of redundancy further in specific implementations. Multi-core architectures with Voltage Frequency Islands enable core sparing to optimize manufacturing chip yield, which asymptotically improves the die yield and cost from the traditional $O(A^3)$ to $O(\sqrt{A})$ with optimally sized cores.

Chapter 5

Network-on-Chip Simulation Infrastructure

One of the key efforts of this work is to demonstrate the way performance gains of adaptive network routing can be used to compensate for on-chip PVT variations. This issue can be studied with analytical modeling to determine the opportunities available from various routing algorithms and methods. However, analysis can only define the bounds for very structured and easy to model cases, such as a uniformly distributed fixed rate traffic pattern. In practice, both spatial and temporal communication patterns of real applications are non-trivial to model, and a simulation provides a more useful mechanism to study the performance of these communication topologies. Additionally, a simulation framework can easily incorporate PVT variations, while the composition of analytical variation models with the network performance models would create a complex, intractable problem that would result in little or no intuitive understanding of the key trends.

This chapter describes the NoC simulation infrastructure that we developed. Although several network simulators were available, such as NS-2 [3] and CMU's cycle stepping NoC simulator [50], none provided the flexibility and configurability to simulate the impact of performance variations. A NoC router comprises several different components such as buffers, crossbars, links, arbiters and router logic. PVT variations affect each component differently depending on its circuit structure, and thus each component must be modeled as an independent *asynchronous* router module with its appropriate variation profile. To gain the flexibility to model a range of implementation and PVT variations scenarios, our own discrete event simulator with performance parametrizable simulation actors that represent NoC router components. Our NoC simulator opened the opportunity to model the router at the micro-architecture level and enabled experiments with routing algorithms—the main focus of this work—as well as with flow control, channel allocation, and switching.

5.1 Simulation Infrastructure

The developed Network-on-Chip simulation infrastructure comprises four principal components illustrated on Figure 5.1: network architecture and communication traffic synthesis, performance space synthesis, discrete event simulator, trace

Network Architecture	Dynamic Processes	Communication Traffic
Topology	Routing Algorithm	Spatial Pattern (app. graph)
Total Node Count	Arbiter Type	Intended Injection Rate
Input Buffer Size	Performance Model	Avg. Packet Length
Virtual Channel Count	(nominal, stochastic,	
Dimension Count	systematic)	

Table 5.1: NoC simulator parameters

and measurement reporting. The simulation parameters outlined in Table 5.1 define its operation.

Network Architecture and Communication Traffic Synthesis instantiates a graph of core/router tiles that represents an on-chip network of the specified topology and size. Each tile is realized as a parametrizable router, flit source and flit sink. The flit source and sink emulate the communication behavior of a processor core adjacent to a router. Section 5.2 describes the details of router micro-architecture, which are defined by the network topology and its dimension, the number of virtual channels, and the flit buffer depth. Section 5.3 details the flit source and flit sink components, which generate the spatial communication traffic patterns based on the application task graph and its mapping onto the set of cores.

Performance Space Synthesis is the process of assigning simulated performance parameters to the simulated components (the actors). The performance model defines the latency and throughput of router components, such as input

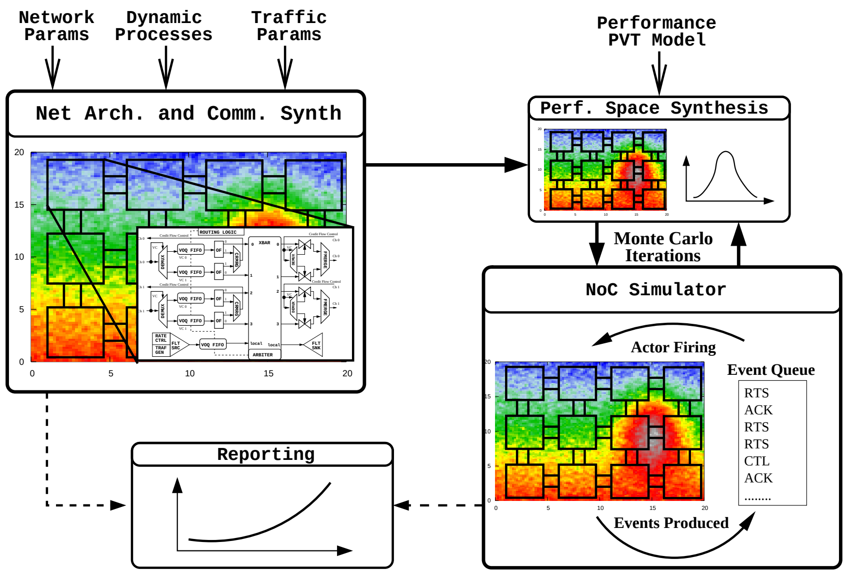


Figure 5.1: NoC Simulator Components

buffers, multiplexers, connection cross bars, inter-router communication channels. Together these components determine the *router* packet transmission (single hop) latency and throughput. The infrastructure allows for dynamic and simulation time dependent changes to the performance model parameters that affect individual router operation. For example, as the task execution activity changes so does the die temperature, which impacts latency and throughput at the tile's operating region. Performance Space Synthesis is implemented as an event-driven actor in the simulation framework, and this permits simulating dynamic voltage and temperature variations as well as run-time system driven DVS/DFS changes.

Figure 5.1 illustrates the NoC simulation flow for a stochastic performance or PVT variation model. To evaluate NoC performance the infrastructure uses a Monte Carlo approach with large number of repeated simulations. The Network-on-Chip topology is synthesized only once per set of architectural parameters. However, the router components and inter-router channels must be re-annotated with generated sample performance parameters for each Monte Carlo iteration.

Discrete Event Simulator is the core of the infrastructure. The simulator is essentially an event dispatch loop, and the event-handling logic particular to simulated router components is the key to its implementation. Sections 5.1.1 and 5.1.2 describe the implementation of the simulator and the key actors.

Trace and Measurement Reporting provides mechanisms to collect, aggregate and visualize network performance statistics that include average packet latency *vs* offered bandwidth, and channel and router utilization.

5.1.1 Discrete Event Simulator

The simulator contains an event queue and an actor graph that represents a graph of NoC routers (more accurately, of the router components), traffic sources and sinks. The actors communicate by exchanging events. An event contains a source and a destination actor in addition to the time-stamp, which orders events in a queue. The simulator iteratively removes an event with the smallest (earliest) time-stamp from the top of the event queue, and activates its *destination* actor, *i.e.* invokes the actor's event handling code. As a result of an activation, the actor may produce new events and post them on the queue.

More formally, an event $e \in E$ contains a time-stamp tuple (t, s), where $t \in \mathbb{R}$ represents simulated time and $s \in \mathbb{I}$ represent a "delta step" at the simulated time t. The tuple rather than a single value t is required for causality in the simulated components, which is particularly critical to establish an event evaluation order for zero-delay components and feedback cycles between actors. An actor can be thought of as a function f that maps an input event e onto a set of output events E_o, while mutating the component's internal state:

$$f : (e, S_f) \mapsto (E_o, S_f^n) \text{ s.t. } \forall_{e_0 \in E_o}(e < e_0) \tag{5.1}$$

where S_f and S_f^n are the current and next actor states. The events form a partial order in the event queue. Given two events $e_0 = (t_0, s_0)$ and $e_1 = (t_1, s_1)$, we define a simple lexicographical partial order in the event queue:

$$(e_0 < e_1) \Rightarrow [(t_0 < t_1) \vee ((t_0 = t_1) \wedge (s_0 < s_1))] \tag{5.2}$$

If the simulator removes events from the queue in order and fires the respective destination actors, the system would accurately simulate the actors' execution semantics and interactions. This does not imply that it would exhibit deterministic simulation behavior, since the determinism depends on the actors' firing logic. For example, the execution semantics of an event merge actor dictate whether it behaves deterministically under all combinations and orderings of input events. This actor can be implemented as a deterministic "round-robin" merge, or the input events may be merged according to their *partial* time order. The *partial* order can be the source of non-deterministic behavior. The actor's interactions and dependencies with adjacent actors in the communication topology dictates whether the simulated system is free of deadlock.

The NoC simulator supports three key events, that help to establish communication flow control between the actors.

- **RTS** (request to send) An actor posts an RTS event to another actor to indicate that it has a data token that is ready to be transmitted. In theory, an actor can post an infinite number of RTS events with the same time stamp, which of course would not accurately represent any physical implementation. Section 5.1.2 describes examples of actors with a limited finite throughput T. Such an actor can send at most T RTS events from a particular output port per unit of time, constrained only by the throughput of its downstream components.

- **ACK** (acknowledgment) An actor responds to a RTS with an ACK event to indicate the time when it is ready to consume a data token. The response can be instantaneous limited only by actor's throughput, or can be delayed indefinitely until appropriate resources are available or an expectant downstream ACK arrives. For example, if an actor is a fixed size FIFO buffer, it can only reply with an ACK if it has a buffer space or when a buffer space becomes available to accommodate the token. This simple handshake matches the production and consumption rates between components and mimics GALS timing.

- **CTL** (control) represents the events that modify actor state out of band or by the simulation environment. For example, a simulation director sends a CTL event to an actor to its throughput in response to on-die temperature change.

5.1.2 Implementation of Key Actors

Actor event handling logic defines simulation functionality and its correctness. An improperly implemented actor can create a deadlock in the simulation, impeding any simulated time progress. An actor can be thought of as an independent thread of execution, an infinite loop that blocks waiting for events and

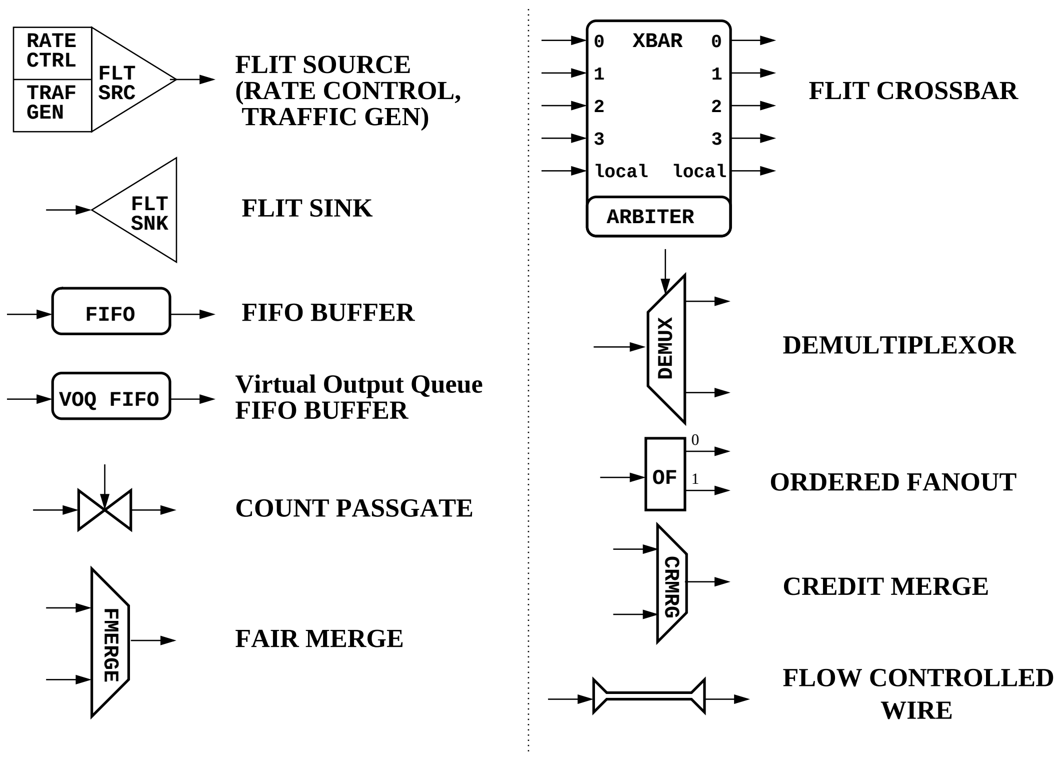


Figure 5.2: NoC simulator components that exchange discrete events.

handles them one at a time. Below we describe the details of the key actors in the simulator, which demonstrate the implementation principles on which the router components are constructed. For simplicity, the description of the `CTL` event handling is omitted as its implementation typically involves only simple mutation of actor's timing or other parameters but not the actor's simulated state. Our graphical symbols for the actors below are shown on Figure 5.2.

Rate Controlled Source Actor

Rate Controlled Source sends a data token to a downstream actor at a maximum rate of $1/t_{tok}$, where t_{tok} is the token transmission time. This actor maintains state variable T_{next} to indicate when it is legal to send the subsequent data token as to not to exceed the maximum throughput.

```
1: Rate Controlled Source Actor($t_{tok}$)
2:
3: {generate next data token to be transmitted}
4: $data \Leftarrow GenerateDataToken()$
5: {start by posting the RTS event with simulated time $(t = 0, s = 0)$}
6: $PostEvent(RTS, data, (0, 0))$
7: $T_{next} \Leftarrow t_{tok}$ {time when next token can be transmitted}
8: loop
9:    {wait for an event $e = (t, s)$ destined for this actor}
10:   $(t, s) \Leftarrow WaitForEvent()$
11:   if $event\_type(t, s) = ACK$ then
12:      {generate next data token to be transmitted}
13:      $data \Leftarrow GenerateDataToken()$
14:      {the earliest time that the next RTS can be posted}
15:      $T_{rts} \Leftarrow \max(t, T_{next})$
16:      {post the event with simulated time $(T_{rts}, s + 1)$ to the queue}
17:      $PostEvent(RTS, data, (T_{rts}, s + 1))$
```

```
18:       {update the earliest time the next token can be sent out}
19:       $T_{next} \Leftarrow T_{rts} + t_{tok}$
20:    end if
21: end loop
```

Algorithm line 15 ensures that the actor does not exceed the specified maximum token rate independent of the rate of downstream actors. The source actor does not handle RTS events. Notice, that each time that an event is posted, delta step is incremented by one to guarantee *total* ordering between dependent events, which is required in a causal system (line 17).

Rate Controlled Sink Actor

Rate Controlled Sink receives a data token from an upstream actor at a maximum rate of $1/t_{tok}$, where t_{tok} is token transmission time. Its operation is quite similar to the Rate Controlled Source.

```
 1: Rate Controlled Sink Actor($t_{tok}$)
 2:
 3: $T_{next} \Leftarrow 0$ {time when next token can be ack'd}
 4: loop
 5:    {wait for an event $e = (t, s)$ destined for this actor}
 6:    $(data, t, s) \Leftarrow WaitForEvent()$
 7:    if $event\_type(t, s) = RTS$ then
 8:       $ProcessDataToken(data)$
 9:       {the earliest time that this token can be acknowledged}
10:       $T_{ack} \Leftarrow \max(t + t_{tok}, T_{next})$
11:       {post the event with simulated time $(T_{ack}, s + 1)$ to the queue}
12:       $PostEvent(ACK, (T_{ack}, s + 1))$
13:       {update the earliest time the next token can be sent out}
14:       $T_{next} \Leftarrow T_{ack} + t_{tok}$
15:    end if
```

```
end loop
```

Flow Control Wire Actor

The Wire Actor simulates a latency insensitive communication channel with a simple 4-step handshake protocol. Its simulated performance is defined by transmission latency l and the token (symbol) time t_{tok}. The Wire Actor receives a RTS from upstream and forwards it to a downstream actor with a simulated time shift of l. Its implementation also sets the maximum token transmission at $1/t_{tok}$. Since this actor has no token buffering, latency and token time are related, *i.e.* the larger of the two values defines its performance.

```
Flow Controlled Wire Actor($l$, $t_{tok}$)

$T_{next} \Leftarrow 0$ {time when next token can be ack'd}
loop
  {wait for an event $e = (t, s)$ destined for this actor}
  $(data, t, s) \Leftarrow WaitForEvent()$

  if $event\_type(t, s) = RTS$ then
    {handle RTS from the upstream actor}

    {compute the time to send downstream RTS}
    $T_{rts} = \max(t + l, T_{next})$
    {post the event to the downstream actor w/ simulated time $(T_{ack}, s + 1)$}
    $PostEvent(RTS, data, (T_{rts}, s + 1))$
    {enforce maximum transmission rate}
    $T_{next} = T_{rts} + t_{tok}$

  else if $event\_type(t, s) = ACK$ then
    {handle ACK from the downstream actor}

    {ACK to the upstream actor w/ simulated time $(t, s + 1)$}
```

```
22:         $PostEvent(ACK, (t, s + 1))$
23:     **end if**
24: **end loop**
```

FIFO Buffer Actor

FIFO buffer enables efficient communication between two heterochronous actors by providing buffering that decouples instanteneous rates of the producer and consumer. This actor can also be used to simulate a wave-pipelined transmission on a wire with multiple values in flight. This actor has three parameters: transmission latency l, token time t_{tok}, buffer capacity S. The latency defines the minimum amount of time between the enquing a token from the input and its dequeuing. The token time parameter defines the maximum input and output token rates.

Since this actor decouples its upstream from its downstream actor, the implementation is significantly more complex than that of the Flow Controlled Wire. This actor has an additional state variable $pending\_RTS$ that stores the last unacknowledged RTS event from the producer. Typically, when an RTS event arrives, it may not be immediately acknowledged if buffer space is not available. The RTS is stored in $pending\_RTS$ until it can be ACK'd when the buffer space becomes available to store the transmitted token. The actor stores a token *data* and its ready time t_r in a fixed size queue, represented as a tuple $(data, t_r)$. Token ready time t_r is defined as token arrival time plus actor's transmission latency l. The actor contains a fixed size token queue that supports the following operations:

- $head(Q)$ returns the token at the head of the queue
- $tail(Q)$ returns the token at the tail
- $enque(Q, d)$ appends token d to the tail
- $deque(Q)$ removes a token from the head of the queue
- $capacity(Q)$ returns the maximum queue capacity defined at actor's construction time
- $size(Q)$ returns the number of tokens in the queue
- $free\_space(Q)$ returns the free space, $capacity(Q) - size(Q)$

The actor operation is described with the following algorithm:

```
FIFO Buffer Actor($l$, $t_{tok}$, $S$)

$Q[S]$ : queue data structure with max capacity $S$
$pending\_RTS \Leftarrow \emptyset$
$T_{next} \Leftarrow 0$ {time when next token can be ack'd}
**loop**
    {wait for an event $e = (t, s)$ destined for this actor}
    $(data, t, s) \Leftarrow WaitForEvent()$

    **if** $event\_type(t, s) = RTS$ **then**
        {handle RTS from the upstream actor}

        **if** $free\_space(Q) > 0$ **then**
```

```
14:         $T_{ack} = \max(t + t_{tok}, T_{next})$
15:         {ACK to the upstream actor w/ simulated time $(T_{ack}, s + 1)$}
16:         $PostEvent(ACK, (T_{ack}, s + 1))$
17:         {add the data to the buffer}
18:         $enque(Q, (data, T_{ack} + l))$
19:         {update the earliest time the next token can be sent out}
20:         $T_{next} \Leftarrow T_{ack} + t_{tok}$
21:
22:         {if this RTS added a token to an empty buffer}
23:         { send an RTS downstream accounting for transmission latency}
24:         if $size(Q) = 1$ then
25:            $PostEvent(RTS, (data, T_{ack} + l, s + 2))$
26:         end if
27:      else
28:         {the buffer is full; save RTS request until it can be ACK'd}
29:         $pending\_RTS \Leftarrow (data, t, s)$
30:      end if
31:
32:   else if $event\_type(t, s) = ACK$ then
33:      {handle ACK from the downstream actor}
34:
35:      {remove the token that has been acknowledged}
36:      $deque(Q)$
37:
38:      {if there is a pending request from upstream, handle it first}
39:      if $pending\_RTS \neq \emptyset$ then
40:         $(data_p, t_p, s_p) \Leftarrow pending\_RTS$
41:         $T_{ack} = \max(t, t_p + t_{tok}, T_{next})$
42:         {ACK to the upstream actor w/ simulated time $(T_{ack}, s + 1)$}
43:         $PostEvent(ACK, (T_{ack}, s_p + 1))$
44:         {add the data to the buffer}
45:         $enque(Q, (data, T_{ack} + l))$
46:         {update the earliest time the next token can be sent out}
47:         $T_{next} \Leftarrow T_{ack} + t_{tok}$
48:         $pending\_RTS \Leftarrow \emptyset$
49:      end if
50:      {if the queue contains any data, send an RTS downstream accounting for
         transmission latency}
51:      if $size(Q) > 0$ then
```

```
52:         {get next data and its "ready time"}
53:         $(data_q, t_q) \Leftarrow head(Q)$
54:         {send an RTS downstream}
55:         $PostEvent(RTS, (data_q, max(t_q, t), s + 1))$
56:       end if
57:     end if
58: end loop
```

Fair Merge Actor

Fair Merge is a starvation-free, nondeterministic merge of input data tokens into a single stream. The tokens are merged in the chronological order by round-robin input acknowledgement scheme. This actor has N input connections to upstream actors and a single output downstream connection. It has infinite throughput and zero token transmission latency. Other rate controlled actors, such as FIFO buffer or Wire, are responsible for accurate performance simulation, but Merge is only responsible for correct control functionality. Since all actors are composable, this separation between timing and control actors dramatically simplifies system implementation.

```
1: Fair Merge Actor($N$)
2:
3: $pending\_RTS[N] \Leftarrow \{\emptyset, \emptyset, \ldots, \emptyset\}$
4: $currInput \Leftarrow (-1)$
5: loop
6:   {wait for an event $e = (t, s)$ destined for this actor}
7:   {here, the event contains the Merge input on which this event arrived}
8:   $(input, data, t, s) \Leftarrow WaitForEvent()$
9:
10:  if $event\_type(t, s) = RTS$ then
11:    {handle RTS from an upstream actor}
```

```
12:
13:       $pending\_RTS[input] \Leftarrow (data, t, s)$
14:       {determine that there are no other forwarded RTS's}
15:       if $(currInput = input) \bigvee \left(\bigwedge_{i \neq input} (pending\_RTS[i] = \emptyset)\right)$ then
16:          $currInput \Leftarrow input$
17:          {forward this RTS to downstream actor}
18:          $PostEvent(RTS, (data, t, s + 1))$
19:       end if
20:    else if $event\_type(t, s) = ACK$ then
21:       {handle ACK from the downstream actor}
22:
23:       {send the acknowledgment back to the currentInput}
24:       $PostEvent(ACK, (t, s + 1)) \Rightarrow SRC[currInput]$
25:       $pending\_RTS[currInput] \Leftarrow \emptyset$
26:       {find next input with a pending RTS with round-robin}
27:       for $j \in [1, N - 1]$ do
28:          if $pending\_RTS[(currInput + j) \mod N] \neq \emptyset$ then
29:             $currInput \Leftarrow (currInput + j) \mod N$
30:             {forward this RTS to downstream actor}
31:             $(data_0, t_0, s_0) \Leftarrow pending\_RTS[currInput]$
32:             $PostEvent(RTS, (data_0, max(t, t_0), max(s, s_0) + 1))$
33:             RETURN
34:          end if
35:       end for
36:       {if no input with pending RTS was found, do not post any events}
37:    end if
38: end loop
```

5.2 NoC Router Micro-architecture

5.2.1 Router-specific Actors

Figure 5.2 shows discrete event actors implemented to construct our NoC router simulation model. The previously described basic actors can be composed to construct any required router component. However, we have chosen to implement more complex actors directly in C++ to improve simulator efficiency, flexibility and observability, instead of building them by composition. The complex actors are customized for routing and operate directly on network flits[1]. Due to their complexity, below we highlight the salient features and operation details of these actors rather than presenting their detailed event handling code.

Credit Merge

Credit Merge is a specialization of the Fair Merge actor designed to operate on value-less tokens, *i.e.* "token presence" indicator only. This actor is a part of the credit based inter-router flow control. It merges the credits that represent the available input flit buffer slots from different virtual channels and sends the credits back to the source router.

Ordered Fanout

This actor logically simulates a simple, no delay event fanout. The RTS event on the input is replicated to all actor's outputs. The fanout semantics in a data-flow environment require that all output actors acknowledge the RTS events before the fanout actor can acknowledge the RTS on its input. In other words, the

[1] *Flit* is a term for network transmission flow control unit.

actor on the input of the fanout cannot proceed until all fanout destinations have acknowledged the receipt of the token.

The *ordered* fanout is a variant of the fanout actor that communicates with its outputs in order. When an RTS event is received on the input, the actor first sends an RTS to output 0 and waits for an acknowledgment. Only after the output 0 returns an ACK, will the actor send an RTS to output 1 and wait for ACK, and so on for all remaining outputs. Only when the last output has acknowledged the receipt of the token, will the ordered fanout actor send ACK to its input. This scheme guarantees that the output i consumes the data *before* output j, if $i < j$. The Ordered Fanout actor is a part of the credit based flow control mechanism, where it ensures that the crossbar has consumed a flit and thus released input buffer space before flow control credit is generated.

Count Passgate

This actor is a zero-delay passgate the has one data input, one data output, and a special input that controls a counter. A counter can be initialized to an arbitrary non-negative value. The actor passes tokens from the data input to the output as long as the value of the counter is greater than zero. As the token is transmitted through the passgate, the value of the counter is decremented by one. Only an event directed to the special control input increments the counter.

Count Passgate is the key source-side component of the credit based control flow scheme. It is initialized to the input buffer capacity of a downstream router. As flits are transmitted onto a wire, the counter is decremented limiting the number of flits in flight. As the downstream router empties its buffer, it sends credits to allow more flits to be transmitted and increments the value of the counter.

Demultiplexer

This actor has one data input and up to N data outputs in addition to a special control input that specifies the output direction. It forwards the input event to the output specified by the token value on the control input. The primary use for this actor is to forward flits to the appropriate buffer based on their virtual channel association. The actor simulates no delay.

Virtual Output Queue FIFO Buffer

Section 3.2 discusses the need for Virtual Output Queues (VOQs) to solve Head-of-Queue blocking problem in the input queue routers. Figure 5.3 shows the functionality and logical implementation of our VOQ FIFO buffer actor. From the inter-router channel, the flits arrive into an ordinary FIFO buffer, where they are considered by the routing algorithm in order. For each packet header flit, the router computes the output channel, with which the header and the rest of the

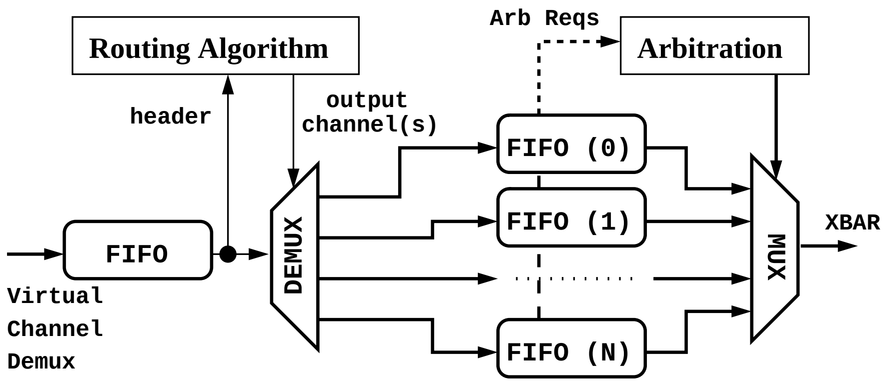


Figure 5.3: Logical implementation of a Virtual Output Queue FIFO buffer.

packet flits are demultiplexed to an appropriate output channel FIFO(i) buffer. The cross-bar arbiter considers all the packets at the head of each virtual output channel FIFO simultaneously and determines which output channels can be granted. The sum of all FIFO capacities is the total capacity of the VOQ FIFO buffer actor. This is only a logical implementation, as in practice, all FIFOs are implemented with a single multi-port circular buffer with complex control. The arbiter may not be able to consider all output channel requests simultaneously, which requires a hierarchical, multistage approach to make the arbiter implementation efficient [80].

In our implementation, an adaptive routing algorithm computes a *set* of output channels for each packet, instead of a single channel. This allows the arbiter to select the output channel after the packet arbitration request and to adapt to the most current congestion conditions by considering several potential VOQs for each packet. An alternative implementation would require the routing algorithm to compute only a single output channel based on the traffic conditions during the routing stage — a long time before the packet makes its crossbar arbitration request. In practice, both approaches are equivalent as long as the input flit buffers are relatively small, and changes in network load balance occur slower than the average packet queue waiting time.

Flit Crossbar and Arbiter

This is the most complex actor in the NoC router simulation, which consists of a passive fully-connected crossbar and an arbiter module that controls when the connections are setup and dismantled. The crossbar operation is simple. Once the arbiter sets up the connection between an input and an output, the crossbar allows a sequence of flits to flow starting from the packet header flit and ending with a packet tail flit. Once the tail flit leaves the crossbar, the connection is severed, and the arbiter is notified.

Since the efficient arbiter implementation was not the subject of this research, our algorithm considers *all* VOQ output channel requests simultaneously, rather than hierarchically. The arbiter attempts to maximally match the VOQs with the output channel by solving a Maximum Weighted Matching (MWM) Problem, which ensures full crossbar utilization. The arbiter is activated by one of the following events. (1) A routing algorithm computed a set of output channels for a packet header flit at the head of the VOQ, and the packet is ready for arbitration. (2) A packet tail flit has exited the crossbar switch, and the corresponding crossbar output is now available.

Routing Algorithm

This actor offers a simple event-driven interface. It receives an event with a routing request from the VOQ FIFO buffer when a packet header flit arrives, and responds with an event with a *set* of output channels. The set of output channels is later considered by the crossbar arbiter and a single output channel is selected and granted for the packet.

The actor contains the routing logic, such as Dimension Order or Minimal Adaptive dual virtual channel algorithms. They determine the output of the actor and, of course, the network performance. The routing algorithms are the subject of Chapters 6 and 7.

Flit Source Actor

This actor is a specialization of the Rate Controlled Source Actor, customized to operate on sequences of flits. The actor simulates the traffic produced by the processor core that accompanies the router. The Flit Source contains injection rate control that defines the upper bound $r \in [0, 1]$ of the injection rate, the fraction of the total ingress[2] bandwidth that the actor may utilize. The Flit Source also contains the configurable traffic generation module that synthesizes packets given a spatial and temporal network communication properties. Section 5.3 discusses traffic generation in detail.

Flit Sink Actor

The Flit Sink actor is a simple specialization of the Rate Controlled Sink Actor, customized to operate on sequences of flits. The actor's primary purpose is to remove flits from the network once they reach their destination. The actor plays a critical role in reporting and statistics gathering infrastructure.

5.2.2 Router Implementation

Due to its modularity, our NoC simulation infrastructure supports a variety of network topologies and router implementations. However, each topology requires specific routing algorithms and deadlock avoidance measures. This work focuses on the two-dimensional mesh of processor cores. Figure 5.4 illustrates a NoC router micro-architecture composed from the discrete event actors. For presentation simplicity, the router as shown supports a one-dimensional network

[2] *Ingress* is the port from the core to the router. *Outgress* is the router to core port.

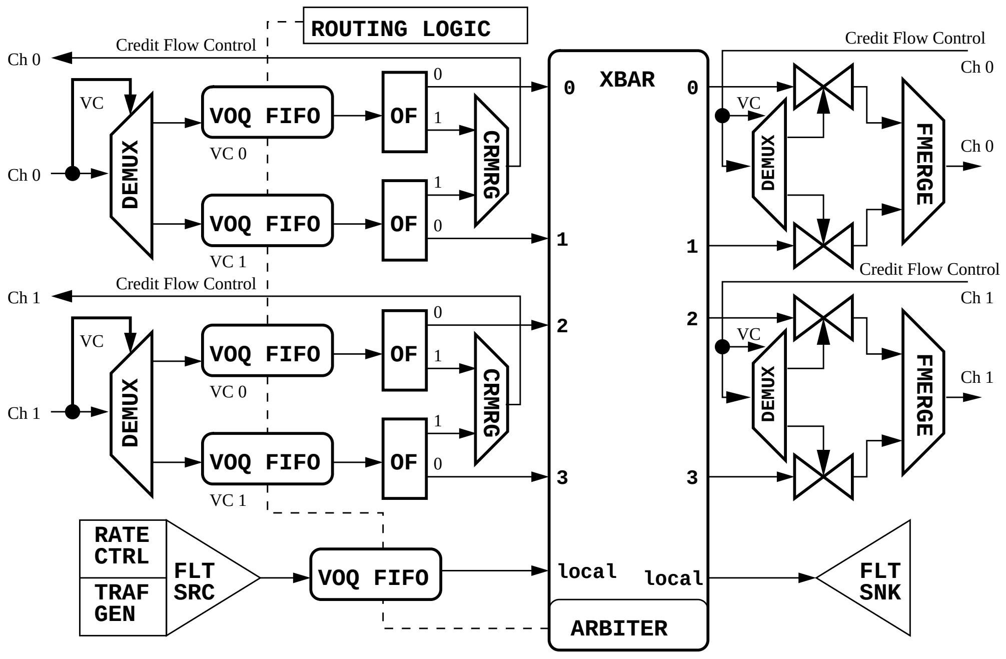


Figure 5.4: Router micro-architecture implemented in NoC simulator.

topology with two duplex channels for positive and negative routing directions. The router contains two virtual channels with credit based flow control. The presented micro-architecture naturally expands to higher dimensional topologies, *e.g.* our 2D mesh, or a greater number of virtual channels, which would be required to implement deadlock-free routing in a torus. The Flit Source and Sink actors simulate the communication behavior of the accompanying processor core.

Consider the following packet routing scenario. A packet header flit arrives on the physical channel input 0, and it is demultiplexed into VOQ FIFO buffer corresponding to its virtual channel 1. As soon as the header flit is stored in the buffer, the VOQ FIFO buffer sends a routing request to the Routing Logic and waits. The Routing Logic responds with a set of valid output channels to the VOQ FIFO actor, which assigns the set to the header flit. Notice that while the router is computing the output channels, some of the subsequent flits belonging to our packet may have arrived and were stored together in VOQ FIFO buffer, available buffer space permitting. The header flit with the valid output channel set is marked "ready for arbitration," and an arbitration request is posted. The arbiter responds when one of the requested outputs is available and sets up the cross-bar connection between the VOQ FIFO (technically, the Ordered Fanout Actor) and the output. Once the connection is established, it will persist until all packet flits travel through the crossbar and are sent through the router output. As the flits are removed one by one from the VOQ FIFO buffer, the Ordered Fanout actor ensures that the crossbar consumes the flit and frees a VOQ FIFO buffer slot before the flow control credit is generated and sent to the Credit Merge actor. The Credit Merge actor aggregates the available buffer slots from all virtual channels in the physical channel 0, and notifies the source router.

The crossbar output is connected to Count Passgates that implement inter-router credit based flow control for each virtual channel. The Count Passgates are initialized to the input VOQ FIFO buffer capacity, and they restrict the number of flits transmitted to ensure that they can be safely buffered downstream without a deadlock. As credits arrive from the downstream router, they increment the Count Passgate, which resumes transmission. A Fair Merge actor combines the flits from two virtual channels and sends them out to the adjacent router. Notice that the Fair Merge actor guarantees starvation freedom and fairness between the virtual channels, *i.e.* assuming both channels have enough credits to transmit, they both receive equal output bandwidth. At the same time, shall one of the virtual channels stall waiting for a credit, the Fair Merge will allow the other channel to consume the entire output bandwidth.

5.2.3 Router Performance Annotation

Figure 5.4 presents the router micro-architecture that comprises communicating discrete event actors. Each actor is a router component whose performance can be parametrized independently in our NoC simulator. For actor implementation simplicity and to ensure its correct operation, each control actor is implemented to simulate the ideal performance, in other words, infinite throughput and zero transmission latency. The router micro-architecture contains the following control actors: Credit Merge, Ordered Fanout, Count Passgate, Demultiplexer, Crossbar and Arbiter, which were described in the earlier sections. The control actor implementation does not have to consider the simulated time. The sole function of a control actor is to ensure the ordered exchange of events with the adjacent components.

The simulation time aware actors determine the router performance, its throughput and latency. These actors include Virtual Output Queue FIFO Buffer, Routing Algorithm Actor, Flit Source Actor, Flit Sink Actor, and Flow Control Wire Actor that implements inter-router communication channels. These actors are parametrized with throughput as token transmission time t_{tok} and transmission latency l as described in Section 5.1.2. Section 5.1 describes one of the key components of our NoC simulation infrastructure: Performance Space Synthesis, which forms a part of the Monte Carlo simulation loop. Given a PVT variation model, which is a combination of stochastic location independent performance distribution and systematic die location aware effects, Performance Space Synthesis annotates each of the router components (*e.g.* FIFO buffer) with a token transmission time and latency.

In our router micro-architecture, the minimum simulated throughput of the Virtual Output Queue FIFO Buffer and Flow Control Wire actors determines the maximum *router* flit throughput. The sum of the latencies of these actors determines the minimal flit transmission time through the router. These bounds, of course, assume no contention or back-pressure from the downstream router. The router throughput and latency are affected by Routing Algorithm Actor parameters, which determine the rate at which the set of output channels can be computed for each packet head flit. If the packet stream comes from or to the local processor core, then the throughput parameters of Flit Source and Flit Sink actors determine the maximum flit throughput.

The router parametrization scheme described here is simple, but it allows us to combine the performance profiles of multiple router components into input buffer and inter-router channel throughput only. This reduction in the number of parameters and focus only on the simulation time aware actors simplifies the PVT variation modeling modeling and helps to identify the critical correlations between router and overall network performance.

5.3 Flit Source and Traffic Generators

To keep our software NoC simulator tractable and efficient, the framework does not simulate the processor execution — all produced traffic is synthetic. Flit Source and Sink Actors attached to the local crossbar ports simulate the network communication behavior of the processor core that accompanies the NoC router. The operation of the Flit Sink Actor is trivial, as its name suggests the actor simply consumes flits at the maximum rate specified by the system performance model and reports the average packet latency and other statistics.

The operation of the Flit Source Actor is defined by the Injection Rate Control and Traffic Generator sub-modules. The Injection Rate Control specifies the upper bound $r \in [0, 1]$ on the fraction of the total ingress bandwidth that the Flit Source can consume. This upper bound rate may not be reached in practice if it exceeds the network saturation rate. For example, assume a uniform random destination traffic pattern and equal packet production rates for all N network nodes. Then the node ingress saturation rate cannot exceed $2/N$ of the total network saturation rate, independent of the specified upper bound. The network topology cross-section bandwidth determines the saturation bandwidth. In uniform random traffic, only half of the nodes communicate to their peers on the other side of the cross-section, which explains the factor of 2 in the bound.

This section focuses on the Traffic Generator sub-module, which is a configurable entity that synthesizes spatial traffic patterns with arbitrary properties. For every network node, the module needs spatial and temporal characteristics of the simulated communication pattern. The spatial characteristics include a set of destination cores and the communication frequencies to each destination. In other words, if the communication pattern is a graph of tasks with fixed on-chip locations, then the traffic generator needs to know the outgoing communication edges.

Temporal communication characteristics, such as burstness, can be captured with detailed packet production times for different cores. They depend on the application implementation, inter-dependencies and mutual synchronization of the tasks. To accurately recreate temporal profiles, the framework must simulate execution of all scheduled tasks and communication flows. Even a seemingly minor change in the task schedule and mapping can significantly impact the temporal communication profile. This is not feasible with a software simulator for more than a few simulated cores.

The inter-task communication *dependencies* can be replicated with a help of static and/or dynamic code analysis and tracing, and as long as imprecise timing is permitted, they can be simulated without processor execution. The effects of caching and network congestion on the traced platform may not be repeatable, however. Our framework only simulates the network operation without the details of processor execution. It does not reconstruct the temporal task dependencies but only focuses on spatial communication patterns and steady state temporal behavior. The simulated network routers contain flit buffers, which to a large degree eliminate the effects of burstness.

The Traffic Generator sub-module can produce artificial spatial communication patterns described in Table 8.1, such as the nearest-neighbor or tornado, or can produce the patterns that closely mimic communication topology of real multi-tasking applications. Artificial communication patterns typically capture a single type of load (im)balance and communication locality. However, real applications have communication patterns that are significantly more complex both spatially and temporary. Real applications have communication behavior that combines several simpler artificial communication patterns, and simulating these patterns has an added advantage that it exercises a range of corner cases. For example, consider two High Performance Computing (HPC) kernels used in this work: GTC and SuperLU (Table 5.2). They are dominated by local communication between the tasks, but also contain all-to-one and one-to-all patterns that create a congested region around the gather sink and scatter source nodes in the network. There is no simple and reliable way to measure the way a particular routing algorithm performs without a complete simulation of the real application communication pattern.

The Traffic Generator sub-module synthesizes traffic based on Message Passing Interface (MPI) application execution traces. Our traces were obtained with the Integrated Performance Modeling (IPM) [2] application profiling layer that non-invasively gathers the communication characteristics of parallel codes as they run in a multi-processor environment. The IPM output does not contain the internal task dependencies that are required to reconstruct or emulate exact timing, but they capture the spatial communication patterns and relative frequencies accu-

Name	Discipline	Problem and Method	Structure
Cactus [6]	Astrophysics	Einstein's Theory of GR via Finite Differencing	Grid
LBMHD [90]	Plasma Physics	Magneto-Hydrodynamics via Lattice Boltzmann	Lattice/Grid
GTC [88]	Magnetic Fusion	Vlasov-Poisson Equation via Particle in Cell	Particle/Grid
SuperLU [87]	Linear Algebra	Sparse Solve via LU Decomposition	Sparse Matrix
PMEMD [32]	Life Sciences	Molecular Dynamics via Particle Mesh Ewald	Particle
PARATEC [24]	Material Science	Density Functional Theory via FFT	Fourier/Grid
FVCAM [105]	Climate Modeling	Atmospheric Circulation via Finite Volume	Grid
MADbench [21]	Cosmology	Cosmic Microwave Background Analysis via Newton-Raphson	Dense Matrix

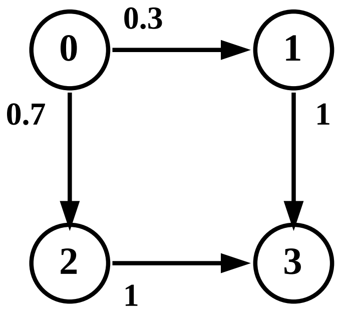
Table 5.2: MPI task communication graphs [120].

rately. As Chapter 6 shows, these traces are sufficient to learn a great deal about the application network behavior and performance.

Figure 5.5 shows a task netlist format extracted from the IPM trace. The netlist contains a set of communication destinations and relative communication frequencies for every task. For example, node 0 sends 30% of output traffic to node 1, and 70% of traffic to node 2. The Traffic Generator sub-module can simulate the specified traffic distribution with the following procedure. Each source-destination flow is assigned a number of tokens proportional to its fraction of consumed output bandwidth. The traffic generator randomly selects a token out of the common

```
0:  (1) @ 0.3, (2) @ 0.7
1:  (3) @ 1
2:  (3) @ 1
```

(a) Netlist description



(b) Corresponding graph

Figure 5.5: An example of the netlist format extracted from IPM trace.

pool and generates a packet to the destination that owns the selected token. The destinations with greater number of tokens have a higher probability of being selected and receive a greater fraction of network packets.

5.4 Task to core mapping

Task-to-core mapping does not qualitatively affect the network performance results in our routing algorithm experiments with application driven spatial traffic generators. Yet in practice, mapping quality has a tremendous quantitative impact on the results. Section 7.5 presents a detailed analysis of the way this mapping affects the network performance improvements from adaptive *vs* oblivious routing algorithms. Our NoC simulation infrastructure uses VPR FPGA placement tool to map the tasks onto the processor cores [16]. The VPR was designed to map a netlist of look-up tables on the Manhattan array of configurable logic blocks

(CLBs) in an FPGA. We represent each task in an HPC kernel graph as a look-up table, and each core/router tile — as a CLB. The VPR computes a "placement" using a simulated annealing algorithm with the bounding box cost function, which attempts to minimize average communication distance between the tasks and preserves locality. After mapping, each core contains a single task.

5.5 Summary

This chapter presents the organization and implementation details of our Network-on-Chip simulation infrastructure that consists of four key components: architecture and communication traffic synthesis, performance space synthesis, discrete event simulator, trace and measurement reporting. The infrastructure is configurable with NoC topology parameters, packet routing and switching algorithms, and PVT variation performance profiles. The actors in discrete event simulator implement various NoC router components and allow realistic modeling of the impact of PVT variations on different types of circuits that comprise the router implementation: memory buffers, interconnect and random logic. A micro-architecture of a configurable mesh/torus multi-virtual channel router is presented to illustrate salient implementation features.

The NoC simulation infrastructure does not simulate task execution in a multi-core system. Instead, it generates a network packet traffic that closely mimics the spatial communication pattern and temporal steady state behavior of real applications. The traffic is generated based on the MPI traces collected from the execution of HPC multi-tasking applications on a super-computer. The ability to perform experiments with realistic in addition to the synthetic and regular traffic patterns strengthens the results discussed in the subsequent chapters by exposing various corner cases.

Chapter 6

Adaptive Routing Algorithms

The heterogeneous performance multi-core architecture described in Chapter 3 offers a great degree of flexibility for run-time application mapping. An ideal mapping would ensure that each computational core and its accompanying network segment are maximally utilized. This chapter focuses on the key resource management task for Network-on-Chip: routing. The application and router performance has traditionally been described with network topology, routing algorithm and switching algorithm bounds. Although these bounds seem to only capture the network properties, in practice application performance depends on the match between the application communication pattern and the underlying network.

Although the analytical topology and routing bounds can be used to estimate network performance opportunities, only an algorithm implementation can identify the realizable gains and demonstrate conclusively the value of various heuristics and communication patterns. This chapter develops several adaptive routing algorithms and evaluates them with our NoC simulation framework. We investigate Dimension Order, Minimal Adaptive, and West First Minimal routing algorithms. The first one serves as a reference performance point, because Dimension Order is trivial to implement in a fixed topology but provides no adaptability to dynamic network conditions. The other two algorithms provide different degrees of adaptability. The simulations identify our Minimal Adaptive Total Congestion (MATC) algorithm as the overall performance winner that we use in the experiments presented in Chapter 7 to evaluate the network performance impact of PVT variations.

6.1 Network performance

The performance of a communication traffic pattern mapped on a network topology can be described by the classic average packet latency *vs* consumed bandwidth curve shown on Figure 6.1. Packet latency is the amount of time for a packet to traverse a path from its source to its destination node. Analytical modeling of such a system is non-trivial, typically requiring a number of important simplifying assumptions about packet arrival rates and a symmetric, regular communication pattern [14, 71]. One of the standard network modeling techniques treats a router input buffer as M/D/1 queuing system. Such a system comprises Markov arrival process (*i.e.* Poisson) of rate λ, Deterministic service process defined by fixed input buffer throughput and one server. By simplifying the packet arrival as Poisson process, the M/D/1 packet queuing delay can be expressed as [123]:

$$Q(\lambda) = \frac{1}{2(1-\lambda)} \tag{6.1}$$

Assume the packet makes on average m router hops from its source to its destination, then the total packet latency can be approximated as

$$L^m(\lambda) = mQ(\lambda) + m \tag{6.2}$$

The packet latency comprises two components: queue wait time and transmission latency. A way to think about packet arrival rate is as traffic injection rate $r \in [0, 1]$ — the fraction of the total network bandwidth that is utilized by the application(s). When all network nodes collectively consume very *small* fraction of the network bandwidth, the expression $L^m(\lambda \approx 0)$ is dominated by term m, which is the *unloaded* packet transmission latency proportional to the average on-chip communication distance. However, as injection rate increases, the queue delay begins to dominate the total packet latency, eventually reaching the bandwidth saturation point where $L^m(\lambda)$ grows to infinity. This simple model clearly identifies the sources and bounds of network performance.

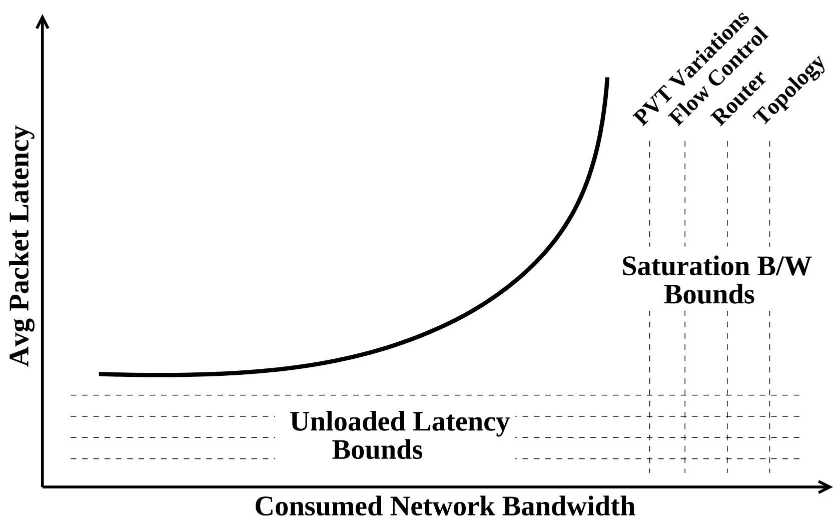


Figure 6.1: Queuing systems defined relationship between packet latency (wait time) and network bandwidth consumption.

Traditionally, researchers identify three types of bounds that constrain the relationship between average packet latency and the consumed network bandwidth (injection rate, from now on): network topology, routing algorithm and flow control (Figure 6.1) [34]. We introduce a new bound imposed by PVT performance variations that represents the network performance degradation that variations create and discuss it in Section 7.1.

6.1.1 Topology Bound

The topology bound on saturation bandwidth is defined as the maximum throughput a network topology can support for a given application communication pattern. This typically depends on the network bi-section bandwidth. For example, consider the uniform random network traffic pattern in a 8x8 2D mesh network, UR(8) in Table 8.1. Due to symmetries in the traffic and the network topology, computing saturation bandwidth bound is simple. Cross-section bandwidth comprises 16 channels, 8 in one direction and 8 in the other. The uniform random traffic implies that half of the traffic from 32 tasks on one side of the NoC bi-section crosses to the other side. Assume for simplicity that the router ingress bandwidth matches that of an inter-router channel. Thus, we obtain that 8 channels are carrying traffic from $32/2 = 16$ tasks, *i.e.* each network node occupies $8/16 = 0.5$ of the critical channel's bandwidth. This corresponds to the network saturation injection rate bound of 0.5, *i.e.* only half of the total network ingress bandwidth can be utilized for the uniform random destination traffic pattern. Similarly, for a regular symmetrical traffic, topology latency bound can also be computed based on the average number of hops that a packet travels. Section 8.1 extends these principles and demonstrates a way to efficiently estimate topology bounds for an arbitrary, non-symmetrical application traffic pattern.

6.1.2 Routing Bound

The routing bound is an improvement on the topology bound. A topology bound assumes that *all* channels are available to carry the application traffic. In contrast, a routing algorithm defines a set of *valid* paths and the corresponding

NoC routers and channels that a packet can traverse for each communicating source and destination. For a simple algorithm, such as Dimension Order routing (Section 6.3), each source and destination pair has a unique routing path. However, adaptive algorithms may offer larger path diversity to the application traffic and thus their bound can approach closer to the topology bound. Computing a routing bound is only simple with load-balanced symmetric traffic patterns, such as uniform random destination. Section 8.2 describes a method to efficiently compute a routing bound on the saturation bandwidth for an arbitrary communication traffic pattern and a routing algorithm.

6.1.3 Flow Control Bound

The flow control bound reflects the router switch bandwidth utilization, which is mostly determined by the switch, arbiter, and router implementation overheads, the flow control granularity, flit and packet sizes. For example, consider a simple switch that suffers from bandwidth degradation due to Head-of-Queue blocking *vs* a more complex switch with Virtual Output Queues [93]. The latter can achieve 100% switch utilization with the ideal Maximum Weighted Matching (MWM) arbitration scheme [96, 33]. In practice, the switch efficiency is limited by a heuristic that can be efficiently implemented in hardware to approximate MWM [45].

6.2 Communication Patterns

A routing algorithm delivers packets from sources to destinations and attempts to balance the packet load across all available network communication channels. To understand its performance, one must understand spatial and temporal communication characteristics of an application. The spatial pattern refers to the network routers and channels that carry the load for each communicating application task pair. The spatial pattern can be characterized by communication locality and load balance. Locality is a measure of the average distance that a packet travels. Intuitively, a task graph with more local communication reduces overall physical resource requirement on the network, and thus may yield lower packet latency and higher network saturation throughput. The load balance is a measure of the variance of utilization across all network routers or channels. Application task-to-core assignment critically affects the network performance because a good assignment maximizes communication locality and balances packet load across available network resources (Section 7.5).

Temporal communication pattern refers to the steady state behavior, such as the rate at which a processing element injects packets into the network, as well as the transient behavior that includes bursts and oscillations in network traffic volume. An accurate modeling of temporal patterns requires application execution on the multi-core system to capture computation to communication ratios of individual tasks, the inter-task synchronization and partial temporal ordering between packets.

This work focuses on spatial communication characteristics and reduces temporal steady state behavior to an average injection rate as is typical in the routing algorithm research. Capturing the exact spatial communication pattern is nontrivial as well, since it is a product of a NP-hard task-to-core assignment algorithm, which often has erratic and non-monotonic results. Thus, we abstract the communication pattern by communication locality and investigate the correlations between the locality and application performance on a given network topology.

Table 5.2 presented the application communication patterns studied in this work. They were extracted from from MPI multi-tasking High Performance Computing kernels. These task communication graphs form a collection of communication patterns from local to global, and their properties are summarized in Table 6.1 with the application size N and the average number of edges per node (degree) d_{avg}. The VPR simulated annealing algorithm was used to map the application tasks to processor cores on a 2D mesh network topology (Section 5.4). Given an application task graph, the mapping algorithm strives to minimize the average communication distance. Table 6.1 partitions our set of benchmarks by size into groups of 64 and 256 nodes, and within their group the applications are

Name	D_{avg}	N	d_{avg}
gtc3-64	1.09	64	2.2
cactus-64	1.73	64	4.5
fvcam_2d	2.62	64	14.4
lbmhd-64	3.37	64	6
pmemd-64	5.24	64	63
gtc2	1.12	256	3.1
mdh2d	2.02	256	4
cactus-256	2.12	256	5
madbench1	3.31	256	13.3
slu-256	5.72	256	30.9
lbmhd	5.81	256	6
madbench2	5.83	256	39.1
paratec-256	10.6	256	255

Table 6.1: A summary of HPC kernels grouped by size and sorted by communication locality.

sorted by the average communication distance (D_{avg}) — the number of router hops that a packet travels. Notice, that our benchmarks represents a range of localities from a nearest neighbor communication pattern (*e.g.* `gtc3-64`) to a global communication pattern (*e.g.* `paratec-256`). This and the following chapters investigate the correlation between application communication locality and its network performance.

6.3 Routing Algorithms

The topology and router bounds can be used to estimate the network performance for relatively simple and symmetric communication patterns. However, only a routing algorithm implementation can identify the realizable gains and demonstrate conclusively the value of various heuristics. We discuss several routing algorithms: Dimension Order, Minimal Adaptive, and West First Minimal Router. The first serves as a reference performance point, because Dimension Order is trivial to implement in a fixed topology but provides no adaptability to dynamic network conditions. The other two algorithms provide different degrees of adaptability, and they are minimal, which guarantees live-lock free operation. We evaluate the variants of these adaptive algorithms to identify the overall winner across our set of HPC benchmarks. We use the best algorithm to study the impact of PVT router and channel parametric variations on the network performance.

This work considers several output channel selection heuristics that use congestion metrics to make adaptive decisions. While they vary in the implementation complexity, no single heuristic consistently delivers maximum performance gains. Instead, a routing algorithm with the *most adaptive choices* consistently performs the best.

To describe a routing algorithm, we define a common nomenclature. The algorithms below apply to operate in a (n, k)-mesh, where n is the dimensionality and k is the dimension cardinality. They are distributed, *i.e.* each NoC router executes an instance of the algorithm and maintains its own local state, and use the following two special data types `Address` and `Channel`:

- `Address` represents a uniquely identifiable position on the mesh as a vector A with n elements: $A = (a_0, a_1, \ldots, a_{n-1})$, where $\forall i \in [0, n) a_i \in [0, k)$.
- `Channel` represents an input or output router channel, identifiable as a tuple (m, d, v), where $m \in [0, n)$ is the dimension and $d \in \{+, -\}$ is the direction, $v \in \mathbb{N}$ is the virtual channel. Channel may also take on a special value "local" to refer to the channel to/from the adjacent processor core. For example, on a 2D mesh $(1, -, 3)$ refers to the channel along the Y dimension in the negative direction assigned to virtual channel 3. In other words, if this router's address is $A = (a_0, a_1)$, then channel "Y-" connects it to the router at $(a_0, a_1 - 1)$.

A routing algorithm can be thought of as a function with the following three arguments, which is invoked when a packet header arrives into an input queue:

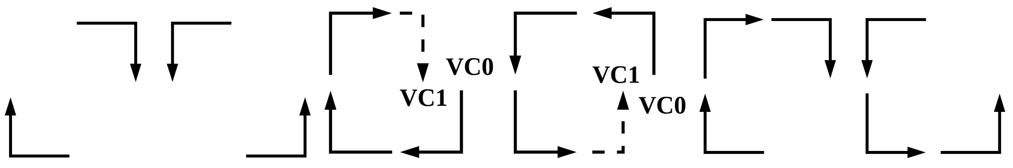
- *InCh* refers to the channel on which the packet has arrived. This is required to disallow appropriate turns in the packet's paths for deadlock-free operation.
- *MyAddr* is the address of the router executing the algorithm.
- *DestAddr* is the packet's destination address.

Upon completion the routing algorithm returns a set of valid output channels and makes the packet ready for crossbar arbitration.

6.3.1 Dimension Order Routing

In spite of its oblivious routing approach, Dimension Order (DO) routing is a simple and popular choice for NoC implementations [135]. The algorithm is naturally deadlock-free, as demonstrated by the valid traffic turns on Figure 6.2(a) that cannot form a cyclical resource dependency. As all minimal routing algorithms, DO is live-lock free as long as the crossbar switch scheduling and arbitration are free of starvation because after every routing decision, a packet is one network hop closer to its destination. Dimension Order routing serves as a reference point to evaluate our adaptive algorithms. It does not adapt to changing network congestion nor does it affect the existing communication load balance. When the traffic is mostly balanced, DO routing works well and delivers the performance that is close to the topology bound. It is popular with system designers due to its low implementation complexity and cost, and because the designers often approximate the expected network traffic as balanced uniform random traffic. The pseudo-code for the algorithm is shown below.

```
1:  DO_ROUTE ($InCh$ : Channel; $MyAddr$, $DestAddr$ : Address) : Channel
2:
3:  # iterate through all dimensions to find the first difference
4:  for all $i \in 0..(n-1)$ do
5:      $\Delta \Leftarrow DestAddr[i] - MyAddr[i]$
6:      if $\Delta > 0$ then
7:          return $(i, +, InCh[v])$
8:      end if
9:      if $\Delta < 0$ then
10:         return $(i, -, InCh[v])$
```



(a) Dimension Order (b) Minimal Adaptive (c) West First

Figure 6.2: Allowed routing turns to avoid cyclical resource dependency

```
11:     end if
12: end for
13: return local
```

6.3.2 Minimal Adaptive Routing

An adaptive routing algorithm can be broken down into two components. The first one computes a set of valid output channels that are consistent with the algorithm's objectives. The second selects from this set to compute a single output channel for the packet. As Section 5.2 describes, we developed a router architecture that splits these two functions, allowing the first component to compute a set of valid outputs as soon as the packet arrives in the input flit queue even if it is not up for output channel arbitration. Then, when the packet is ready for arbitration, the arbiter can select an output channel for each pending packet from its *set* of valid output channels.

This approach complicates arbiter implementation in that it must now consider requests for multiple rather than single output channel in each Virtual Output

Queue [93]. In practice, this is not a significant complication or an impediment to arbiter scalability. Modern arbitration logic already has to compute on multiple simultaneous conflicting requests from different input channels, and a hierarchical implementation would ensure high performance and scalability for 2–3 rather a single candidate as well [80].

A *minimal* adaptive routing algorithm computes a set of output channels that enable forward progress to a packet's destination node. Therefore, the algorithm is live-lock free. Deadlock-freedom is achieved with virtual channels. Figure 6.2(b) shows the two prohibited turns that eliminate cyclical resource dependency in a 2D mesh. Instead of taking a prohibited turn, a packet switches from virtual channel 0 to 1. In a minimal routing algorithm, traffic travels from its source to destination and never takes a mis-routing hop outside of the shortest topological path to its destination (in terms of the number of router hops). Therefore, only Eastbound traffic direction can ever switch the virtual channels. There cannot form a cyclical resource dependency between Eastbound traffic and the traffic in other directions within the same virtual channel, which assures deadlock-free operation.

Procedure $TWODIM\_ASSIGN\_VC$ performs the virtual channel assignment consistent with Figure 6.2(b):

```
1: $TWODIM\_ASSIGN\_VC$ ($InCh$ : Channel; $dest\_dim$: $[0, n)$ ) : integer
2:
```

```
# compute an output virtual channel
if $InCh = local$ then
    return 0
else if $InCh[v] = 1$ then
    return 1  # if it is already in VC=1, then stay there
else if $(dest\_dim = 1) \wedge (InCh[m] = 0) \wedge (InCh[d] =' -')$ then
    return 1  # switch EN and ES turns to VC=1
else
    return 0
end if
```

Using the virtual channel assignment function, we construct the Minimal Adaptive algorithm:

```
MA_ROUTE ($InCh$ : Channel; $MyAddr$, $DestAddr$ : Address) : Channel

$P \Leftarrow \emptyset$
# iterate through all dimensions to find all differences
for all $i \in 0..(n-1)$ do
    $\Delta \Leftarrow DestAddr[i] - MyAddr[i]$
    if $\Delta > 0$ then
        $P \Leftarrow P \cup \{(i, +, TWODIM\_ASSIGN\_VC(InCh, i))\}$
    end if
    if $\Delta < 0$ then
        $P \Leftarrow P \cup \{(i, -, TWODIM\_ASSIGN\_VC(InCh, i))\}$
    end if
end for
if $P = \emptyset$ then
    return $local$
else
    return $SELECT(P)$
end if
```

The algorithm computes all possible output channels along the minimal path, but the key to its adaptivity is $SELECT()$ function that uses a range of "environmental" factors to reduce the set of valid channels into the best one at the moment. We considered several selection functions that can be implemented in a distributed router. The selection function operates within the output channel switch arbitration logic, and thus this function handles simultaneous requests from multiple input buffers.

SELECT_OCF Oldest Channel First is a simple starvation free switching heuristic with low hardware implementation complexity. The algorithm iteratively selects the oldest arbitration request and attempts to assign it to an available output channel until all pending requests have been considered. This greedy approach shows the most benefit when a Virtual Output Queue request contains a large set of output channels ($|P| > 1$ in line 17 of the algorithm above). Intuitively, **OCF** attempts to keep input channel occupation low by preferring the longer waiting requests. This heuristic is a good comparison point to demonstrate the inherent advantages of the greater network path diversity that is exploited by an adaptive router over the oblivious algorithm without a more sophisticated output channel selection criteria.

SELECT_BHTA This heuristic attempts to select the output channel based on a combination of parameters that can be measured locally by the router, including:

- B = backlog length, the total number of flits in the input buffer that are waiting to be transmitted to the specified output channel.

- H = backlog length of the packet currently in transit to the specified output channel
- T = observed output channel throughput. Averaged over a small window, this metric indirectly captures performance and congestion level of the adjacent router.
- A = request age, or the total queue waiting time
- W = weight assigned by a routing algorithm. This can be used by the router to "favor" certain output channels within a computed output channel set P. For example, an adaptive router may favor X direction over Y, and allow the traffic conditions to dictate when to switch between oblivious and adaptive operating modes.

The arbiter with the SELECT_BHTA function assigns a set of input channels (set N) that request connections to a set of output channels (set M) in the crossbar. This problem can be formulated as Maximum Weighted Matching (MWM) on a bipartite graph $G = (N \cup M, E)$, where the edges span the two node sets: $\forall (n, m) \in E \quad n \in N \wedge m \in M$ [31]. Each edge (n, m) represents a pending request from an input buffer n to an output channel m and has the weight of:

$$w = \frac{WA(B + H)}{T^{\alpha}} \tag{6.3}$$

This expression loosely was derived from the length of time that would be required to completely drain the backlog destined to an output channel: $\frac{(B+H)}{T}$. The age A ensures that the Maximum Weighted Match-based algorithm does not starve an input channel. The age A increases with time and makes the corresponding pending request more prominent. The experiments have shown that weighting the observed channel throughput more heavily with $\alpha > 1$ may sometimes improve the overall algorithm performance.

These channel weights are computed for each output channel request, and the solution is obtained with Integer Linear Program MWM solver. The solution provides a valid assignment of input channels to the outputs, which favors draining the flit backlogs with longer queue waiting times.

SELECT_TC The Total Congestion heuristic uses the measurements reported by the adjacent routers to select the minimally congested output channel. Our experiments found the total flit buffer occupancy to be a very effective congestion metric. The variants, such as "Direction Congestion" that counts only those flits on the adjacent router that are headed in the same direction as the packet being routed, are equally effective on average. Intuitively, since an adaptive algorithm attempts to balance the load in the *entire* network in every mesh direction, a direction-specific congestion metric carries the same information as the total router congestion if the routing algorithm is successful. Thus, an adjacent router is either congested or not, and if it is congested, the backlog for all its outputs should be approximately the same. However, Direction Congestion does incur additional implementation complexity. This makes the Total Congestion a better and simpler selection heuristic that we use in this work.

The routers can exchange congestion measurements with dedicated special connections or by overloading the flow control credits to carry this additional information. A similar scheme in [66] suggests that either implementation should not result in a more than 7% area overhead even for routers with very small input buffer sizes. In our implementation, the routers update their neighbors continuously when their input buffer occupancy changes in the similar manner to the way flow control credits are exchanged.

Figure 6.3 illustrates the simulation results for the `PMEMD-64` and `FVCAM_2D` applications for Dimension Order and Minimal Adaptive routing with different $SELECT()$ functions. Table 6.2 contains the network saturation injection rates for all applications examined. For ease of comparison, for each application the table also contains the saturation injection rates normalized to those from the Dimension Order routing, *i.e.* the ratio R^{MA}/R^{DO}. With the exception of `FVCAM_2D`, **MA OCF** shows the least improvement over the **DO** algorithm as compared to other $SELECT()$ heuristics. Nonetheless, **OCF** does highlight the saturation injection rate gains from greater network path diversity, which are particularly

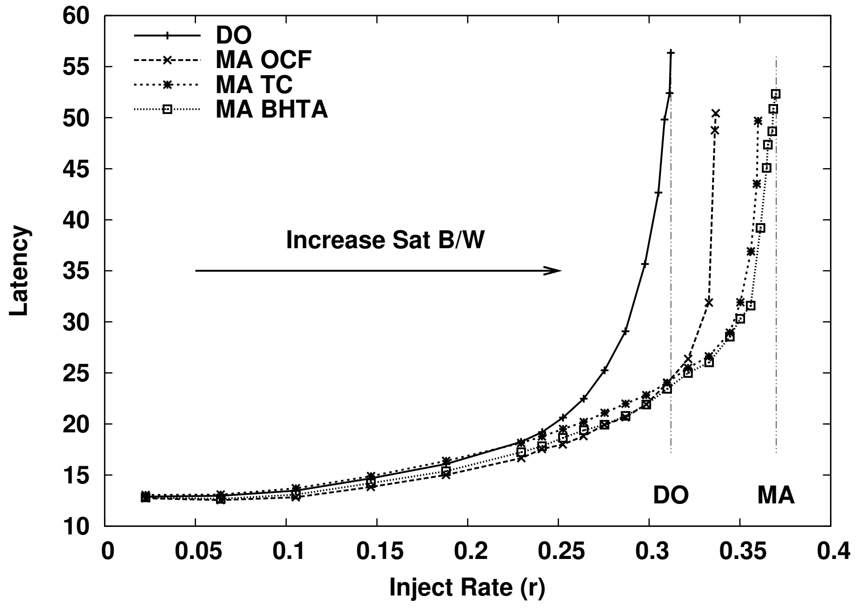
Name	D_{avg}	Saturation Inject Rate (R)				R^{MA}/R^{DO}		
		DO	OCF	TC	BHTA	OCF	TC	BHTA
gtc3-64	1.10	0.96	0.95	0.95	0.95	1.00	1.00	0.99
cactus-64	1.70	0.91	0.89	0.90	0.91	0.98	1.00	1.00
fvcam_2d	2.60	0.53	0.65	0.60	0.63	1.23	1.12	1.18
lbmhd-64	3.40	0.53	0.60	0.61	0.62	1.13	1.16	1.18
pmemd-64	5.20	0.31	0.34	0.36	0.37	1.08	1.15	1.19
gtc2	1.10	0.98	0.98	0.98	0.99	1.00	1.00	1.00
mdh2d	2.00	0.83	0.89	0.89	0.90	1.06	1.07	1.07
cactus-256	2.10	0.79	0.84	0.85	0.85	1.06	1.09	1.08
madbench1	3.30	0.34	0.41	0.41	0.37	1.19	1.20	1.08
slu-256	5.70	0.22	0.24	0.25	0.25	1.12	1.13	1.13
lbmhd	5.80	0.28	0.30	0.34	0.35	1.07	1.21	1.23
madbench2	5.80	0.25	0.25	0.25	0.25	1.00	1.00	1.00

Table 6.2: Performance comparison for different $SELECT()$ implementations.

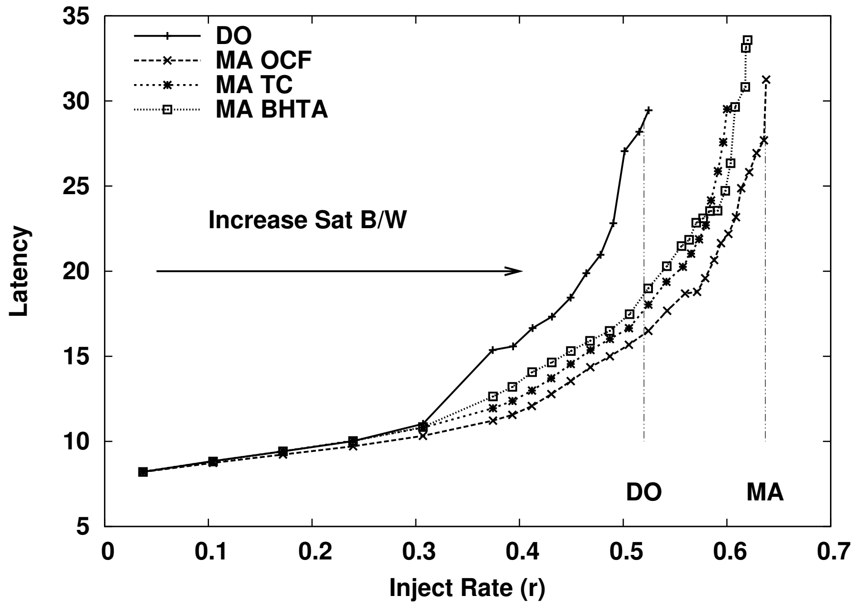
pronounced in the application with mid-range locality, $2 < D_{avg} < 5$. The Total Congestion (**TC**) and **BHTA** algorithms improve the output channel selection further. In spite of the complexity of **BHTA** that takes into account the backlog length, the output channel performance and the packet input buffer waiting time, it only marginally outperforms the Total Congestion (**TC**) heuristic. **TC** is significantly simpler to implement and can be approximated with a local flow control credit counter to further reduce its area overhead.

6.3.3 The Minimal West First Routing

Minimal Adaptive routing discussed in the previous section requires virtual channels to ensure deadlock freedom. Virtual channels complicate input flit buffer



(a) PMEMD-64



(b) FVCAM_2D

Figure 6.3: $SELECT()$ performance impact for two application graphs.

management and flow control, and result in additional area overhead. To evaluate the advantages of our Minimal Adaptive routing with unrestricted path diversity over a simpler deadlock free algorithm that requires no virtual channels, we investigate Minimal West First (MWF) Routing.

A West First algorithm prohibits S→W and N→W turns to eliminate the possibility of a cyclical resource dependency as shown on Figure 6.2(c) [49]. With the restricted turns, the algorithm does not require virtual channels or structured buffer pools, and it might be a preferred option for the resource constrained implementations. On a two-dimensional mesh, the algorithm has a limited adaptivity for west-bound traffic, which first travels only along the west-bound output channels to reach the destination's X column before the router decides on North or South. In effect, depending on an application, MWF may behave very similarly to a Dimension Order router.

Figure 6.4 compares the performance of the Dimension Order (DO), Minimal West First (MWF) and Minimal Adaptive (MA) routing algorithms for `PMEMD-64`. For this application graph, MWF only slightly outperforms the oblivious DO, and its saturation bandwidth does not even compare with the Minimal Adaptive router from the previous section. To further demonstrate the limited adaptivity of MWF router, we designed its two virtual channel version: Dual Minimal West First (DMWF). With DMWF, the east-bound traffic is placed on virtual channel

0 to be routed using the Minimal West First algorithm. However, the west-bound traffic is placed on VC 1 and routed using the mirror Minimal *East* First algorithm. The packets *never* switch between the virtual channels, and thus overall network routing remains deadlock free, while each VC provides maximum adaptivity in both directions. MWF and DMWF algorithms use the **SELECT_TC** function to select an output channel from a set of adaptive choices. Notice that DMWF performs essentially the same as the Minimal Adaptive algorithm for `PMEMD-64` application.

Table 6.3 summarizes application performance with the absolute saturation injection rates, and the normalized saturation rates of the adaptive algorithms to those of the Dimension Order router (R^*/R^{DO}). The main trend observed in the previous sections can be seen in the table: the application graphs with non-local communication patterns reap the most benefit from adaptive routing. The performance gains from the MWF algorithm are significantly smaller than those from the dual virtual channel variant DMWF and the Minimal Adaptive algorithms, primarily due to lack of routing adaptivity for west-bound traffic. DMWF and MA have essentially the same performance in terms of saturation bandwidth, since they offer maximum adaptivity along every minimal packet routing path. DMWF and MA have the same hardware implementation complexity — two VCs in a two-dimensional mesh — but equivalent performance. This work focuses on

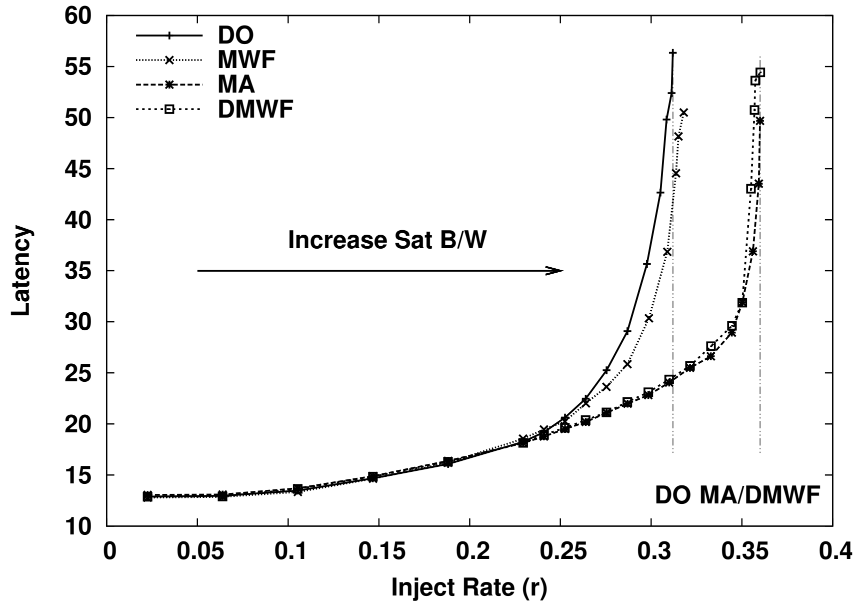
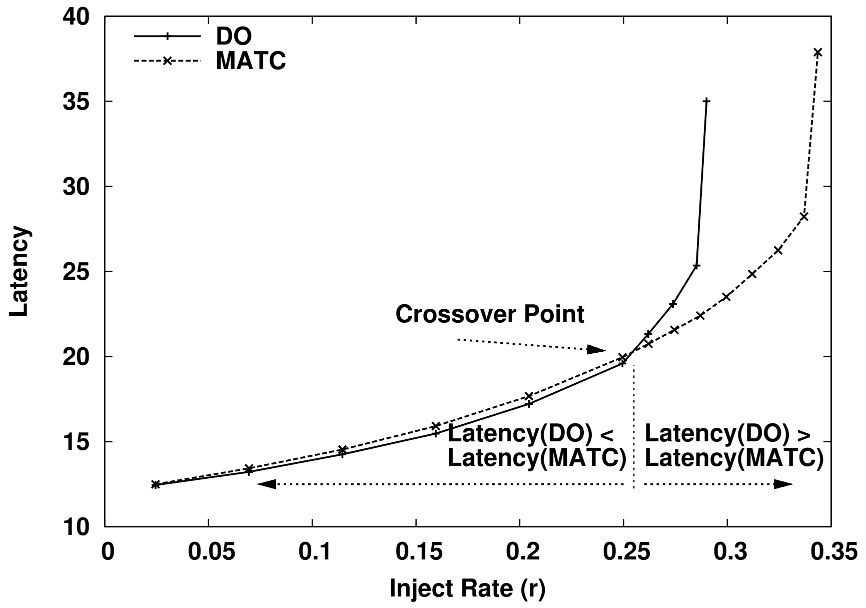


Figure 6.4: Performance of West First algorithms on `PMEMD-64`.

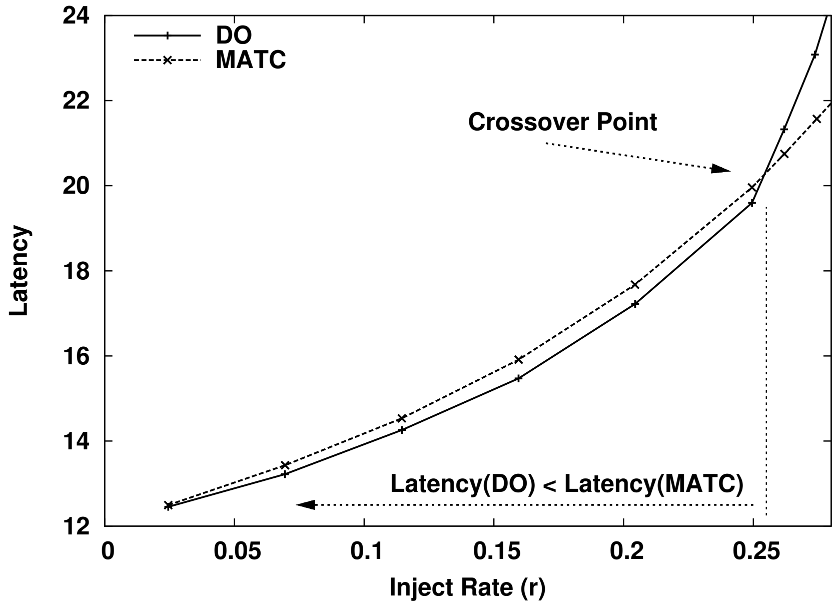
the Minimal Adaptive Total Congestion (MATC) routing algorithm in the experiments.

6.3.4 Latency

So far we have focused only on the network saturation bandwidth and have not considered the average packet latency in the unloaded network. The example on Figure 6.5(a) shows a significant reduction in the saturation bandwidth from the adaptive routing. However, a closer examination of the results reveals that the Minimal Adaptive router also degrades the average packet latency in the low bandwidth utilization region as compared to the oblivious Dimension Order router



(a) Complete curve



(b) Focus on low injection rates

Figure 6.5: `LBMHD` application: the average packet latency for deterministic *vs* adaptive algotithms.

Name	D_{avg}	Saturation Inject Rate (R)				R^{*}/R^{DO}		
		DO	MWF	DMWF	MA	MWF	DMWF	MA
gtc3-64	1.10	0.96	0.96	0.95	0.95	1.00	0.99	1.00
cactus-64	1.70	0.91	0.91	0.90	0.90	1.00	0.99	1.00
fvcam_2d_r1	2.60	0.53	0.56	0.60	0.60	1.04	1.13	1.12
lbmhd-64	3.40	0.53	0.56	0.60	0.61	1.05	1.14	1.16
pmemd-64	5.20	0.31	0.32	0.36	0.36	1.02	1.15	1.15
gtc2	1.10	0.98	0.99	0.98	0.98	1.00	1.00	1.00
mdh2d	2.00	0.83	0.87	0.89	0.89	1.04	1.07	1.07
cactus-256	2.10	0.79	0.84	0.84	0.85	1.06	1.06	1.09
madbench1	3.30	0.34	0.36	0.40	0.41	1.06	1.18	1.20
slu-256	5.70	0.22	0.24	0.24	0.25	1.09	1.13	1.13
lbmhd	5.80	0.28	0.30	0.33	0.34	1.04	1.17	1.21
madbench2	5.80	0.25	0.25	0.30	0.25	1.00	1.20	1.00
paratec-256	10.70	0.19	0.17	0.18	0.19	0.90	0.95	0.99

Table 6.3: Performance of West First algorithms compared to MATC and DO.

(Figure 6.5(b)). This is an example of a very typical behavior of adaptive routing algorithms. When the network is lightly utilized, a routing algorithm should simply send packets along the shortest path and not try to load balance. Unfortunately, the adaptive router selects the least congested output channel based on the buffer utilization of the adjacent routers. At the low traffic injection rates, the buffer utilization is also low, and the congestion metrics are noisy and do not accurately reflect load imbalance. If the Minimal Adaptive algorithm uses noisy metrics in an attempt to balance the communication load across the network, it ends up making bad local routing decisions that degrade packet latency. However, once the injection rate reaches the critical crossover point, the total flit buffer occupancy — our congestion metric — begins to reflect the load imbalance. The adaptive algorithm can most effectively route the packet flows to increase the network saturation bandwidth.

A hybrid approach can deliver the best of worlds: the low packet latency of deterministic algorithms in lightly loaded network conditions, and the high saturation bandwidth of adaptive routing [66, 122]. Such an algorithm operates in the deterministic mode, similar to Dimension Order, when the network bandwidth utilization is low, and switches to an adaptive mode when the utilization is high. The crossover point can be defined with a static threshold or be adjusted adaptively, although the latter has not been extensively studied. The implementation of a hybrid router is not significantly more complex than the implementation of our adaptive algorithms, however, we do not investigate the hybrid routers in this work.

Although the unloaded network packet latency should not be ignored, a number of other factors affect the system response latency to a greater degree than the routing algorithm. The packet transmission latency forms a part of the overall system response time that also comprises cache access latency, congestion and arbitration for off-chip memory access and other system factors. In this work, a 2–4% latency variation induced by the choice of a routing algorithm does not play a significant role.

In contrast, the network saturation bandwidth is the key metric that defines the region where an application can operate productively. Exceeding the saturation bandwidth is akin to page thrashing in virtual memory systems. If a program's working set — the memory space accessed the majority of time — does not fit in the primary memory, the program constantly swaps memory pages with a disk. This thrashing results in a sharp decrease in responsiveness and performance. Saturation bandwidth is the communication system analogue of the working set size. When the application communication requirements exceed the saturation bandwidth, the network becomes unresponsive, and the packet latency increases sharply. At that point, it is no longer possible to extract any computational capacity out of a multi-core system.

6.4 Summary

A network on a chip is a distributed system consisting of interconnected core/router tiles. Its performance and efficient operation depends on a range of factors: design parameters such as topology and cross-section bandwidth, as well as the application that executes on the multi-core NoC system. Spatial and temporal application communication properties such as locality and load balance determine the actual network performance: the saturation bandwidth and the average packet latency. A key performance metric that defines the efficient and practical system operating range for the network is its saturation bandwidth.

We develop several adaptive routing algorithms and output channel selection heuristics and evaluate them with NoC simulation. The experiments show that the Minimal Adaptive Total Congestion (MATC), MA BHTA, and Minimal West First algorithms realize the advantage of adaptive routing over the oblivious routers by taking advantage of the network path diversity. The actual performance results vary from one adaptive algorithm to another and also depend on an application task graph. For example, the Minimal Adaptive algorithms which require two virtual channel for a deadlock free implementation consistently outperform the West First algorithms that requires only one. The Minimal adaptive algorithms can adaptively route the packet traffic in all directions, while West First has a limited adaptability for the west-bound traffic.

Our experiments show that the MATC algorithm combines a simple implementation and the performance close to the router bounds. In the following chapter, we use MATC to evaluate the network performance impact of PVT variations.

Chapter 7

Adaptive Routing and PVT Variations

We present a new PVT variation bound, a theoretical foundation to describe the impact of PVT performance variations on network performance. PVT variations degrade network saturation bandwidth particularly with oblivious routing algorithms such as Dimension Order routing. This chapter focuses on the Minimal Adaptive Total Congestion (MATC) router and demonstrates with NoC simulation that MATC increases the expected network saturation bandwidth and reduces the saturation bandwidth variance as compared to DO on systems affected by PVT performance variations. These gains can be turned into smaller design margins, a boost in power efficiency and performance, in other words, less "over-engineering."

A heterogeneous multi-core architecture of Chapter 3 offers a great degree of flexibility for adaptive routing that strives to ensure that network resources are maximally utilized. To scale with device generations, routing algorithms should not rely on detailed on-chip performance characterization. Instead, they should operate introspectively and adapt to the number of possibly unknown and hard to quantify system performance variables. Our heterogeneous architecture exposes PVT variations as core/router tile performance, which for the communication fabric translates to routers and channels with diverse throughput and latencies. The performance variations upset communication load balance and add to the network congestion. However, an adaptive router can compensate for the variations using only inter-router flow control, because both static and dynamic variation effects are exposed in the same manner as application communication load imbalance and traffic congestion.

7.1 PVT Variation Bound

In addition to the three classical performance bounds that constrain the relationship between the average packet latency and the consumed network bandwidth, we propose a new bound imposed by PVT variations (Figure 6.1). Although the variations create a very complex on-die performance profile, we model them using two simple communication metrics: channel bandwidth and latency. These metrics represent the black-box, outside view of the router, which is consistent with our intro-spective approach to compensate for variations without the detailed performance characterization.

PVT variations turn the nominal channel bandwidth w and latency l from static to random variables. Equation 6.2 shows that the packet latency is simply the sum of packet queue waiting time and the number of routing hops traveled. Let us consider the effect on the packet latency in the unloaded network. Since the inter-router channel transmission latency l can be modeled with a symmetric distribution, the *expected* total packet latency ideally would remain unaffected due to the averaging effect that occurs when individual channel latencies are added up along the packet's path.

Let L be a random variable that represents the average packet latency, *i.e.* it is the sum of individual channel latencies $L = \sum_i^{D_{avg}} l_i$ along the routing path. If l is normally distributed and spatially uncorrelated, which is the case at the router granularity, the standard deviation of L has a diminishing impact relative to the expected value of L as the average communication distance D_{avg} increases [15]:

$$\mu(L) = D_{avg} \times \mu(l) \tag{7.1}$$

$$\sigma(L) = \sqrt{D_{avg}} \times \sigma(l) \tag{7.2}$$

$$\frac{\sigma(L)}{\mu(L)} = \frac{\sqrt{D_{avg}} \times \sigma(l)}{D_{avg} \times \mu(l)} = \frac{1}{\sqrt{D_{avg}}} \times \frac{\sigma(l)}{\mu(l)} \tag{7.3}$$

Thus, any latency degradation is less pronounced in non-local task communication patterns with larger D_{avg}. The packet latency is closer to the expected nominal value $\mu(L)$ as more channel latencies are added together on a longer routing path.

PVT variations affect the the network saturation bandwidth more dramatically. For every packet source and destination, a routing algorithm pushes the traffic through a set of valid network paths. Let us designate this path set as $\mathcal{S} = \{P_i\}$. The *minimal* capacity channel along a path determines its net bandwidth. Considering router performance variations, and assuming for presentation simplicity that $\mathcal{S}$ contains non-intersecting paths, the bandwidth upper bound $w(\mathcal{S})$ through a path set $\mathcal{S}$ can be described as:

$$w(\mathcal{S}) = \sum_{P \in \mathcal{S}} \left[\min_{e \in P} \left(w[e] \right) \right] \tag{7.4}$$

$$\forall P \in \mathcal{S}.\, |P| = D_{avg} \tag{7.5}$$

where set P represents a single path between a source and destination, e is an edge on that path, and $w[e]$ is a random variable that represents the bandwidth through e. The expression optimistically assumes that given path diversity in $\mathcal{S}$, a routing algorithm can and will select the paths with highest throughput. This may not be true, and the limitations of a routing algorithm is the first source of network saturation bandwidth degradation. Further, the path bandwidth $\min_{e \in p} (w[e])$

is the minimum of a set of D_{avg} random numbers $w[e]$. The expected value of this minimum function approaches the lower bound of the distribution of $w[e]$ as the path length D_{avg} increases. This reduction in the effective path bandwidth degrades the overall network saturation bandwidth.

Our analysis demonstrates the advantage of adaptive routing algorithms over the oblivious ones. Since an adaptive routing algorithm can utilize a greater number of paths between every packet source and destination, it accomplishes two objectives. First, it increases the available cross-section bandwidth between a source and a destination, and therefore the router network saturation bound as evident from the summation over a larger path set S in Equation 7.4. Second, and most critically, if you consider all packet paths together in space and time, an adaptive algorithm *aggregates* the bandwidth of each path in $\mathcal{S}$ between a packet source and a destination. This summation *averages* the path bandwidth closer to the nominal value. These properties of adaptive routing are the reasons for improvements in the expected saturation bandwidth and the reduction in its variance, which are discussed in Section 7.2.

7.2 Impact of PVT Variations

The improvement in saturation bandwidth from adaptive routing algorithms can be used to compensate for PVT induced process variations. PVT device (transistor and interconnect) variations manifest themselves as core/router performance. As a result of the variations, a die with nominally homogeneous performance turns into a collection of tiles with heterogeneous performance. For our experiments, we model the variations as router throughput and latency, and measure the network saturation bandwidth for an application. As in Section 7.1, here we assume that router performance is constrained by inter-router channels $(u, v) \in E_N$ with throughput of $C_{(u,v)}$ and latency $1/C_{(u,v)}$. The routers have higher internal throughput than $C_{(u,v)}$ to ensure the full utilization of all input and output channels when the router is not congested. This assumption is true for every well-designed, resource balanced system and represents the optimal NoC router implementation. Abstracting the router performance as channel performance simplifies the analysis of the way the adaptive routing algorithms compensate for the performance variations. This abstraction removes irrelevant details of core and router implementation and enables us to focus on the trends.

To describe the effects of inter-router channel performance variations on the network saturation bandwidth, let us consider the channels in aggregation rather than individually. We define the following terms and variables:

- **Nominal channel throughput C_0.** This static variable represents the throughput (flits per unit of time) of an inter-router channel, which is set by the system designer. In other words, C_0 is an idealized NoC, not impacted by PVT variations. Since our experiments are not tied to a particular implementation, unless otherwise stated we use $C_0 = 1$ in the following discussion.

- **Nominal saturation injection rate $R_0^{\mathcal{R}}$.** This static variable represents the saturation injection rate on the nominal system with a routing algorithm $\mathcal{R}$. In the following discussion, we refer to the saturation injection rates R_0^{MATC} and R_0^{DO} for MATC and DO routing algorithms respectively.

- **Realistic channel throughput C.** This variable represents a channel throughput distribution created by PVT variations. If the distribution is stochastic, C_σ is a *random* variable that can parametrized with the expected value μ and the standard deviation σ. In our discussion, the implied expected value of the channel throughput $\mu = C_0 = 1$

  If the distribution is not stochastic, C is a function that maps on-die location or other factors, such as temperature, onto a value for channel throughput.

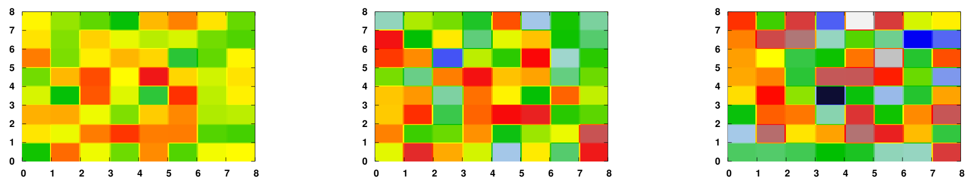
- **Realistic saturation injection rate $R^{\mathcal{R}}_{\sigma}$.** This *random* variable represents a distribution of the network saturation injection rates in a system with channel throughput C_σ and a routing algorithm $\mathcal{R}$. For example, $R^{MATC}_{0.14}$ refers to the distribution of the saturation injection rate obtained on a NoC with channel throughput $C_{0.14}$ — normally distributed as $N(\mu = 1, \sigma = 0.14)$ — and running **MATC** routing algorithm.

  The distribution of the saturation injection rates may itself be characterized by its expected value $\mu(R^{\mathcal{R}}_{0.14})$ and its standard deviation $\sigma(R^{\mathcal{R}}_{0.14})$.

7.2.1 Stochastic Variations

Consider *stochastic* intra-chip performance variations, where each core/router tile has a different communication throughput governed by the PVT variation distribution parameters. In the following experiments, we define the router throughput to be normally distributed across the chip as $C_\sigma = N(\mu = 1, \sigma)$. This is illustrated on Figure 7.1 with density plots.

How do router performance variations affect our key metric, the network saturation injection rate $R^{\mathcal{R}}_{\sigma}$? How can an adaptive routing algorithm compensate for variations? Let us illustrate the effects of performance variations on the `PMEMD-64` application and the **MATC** routing algorithm, which combines a simple implementation and the best performance among the adaptive algorithms we

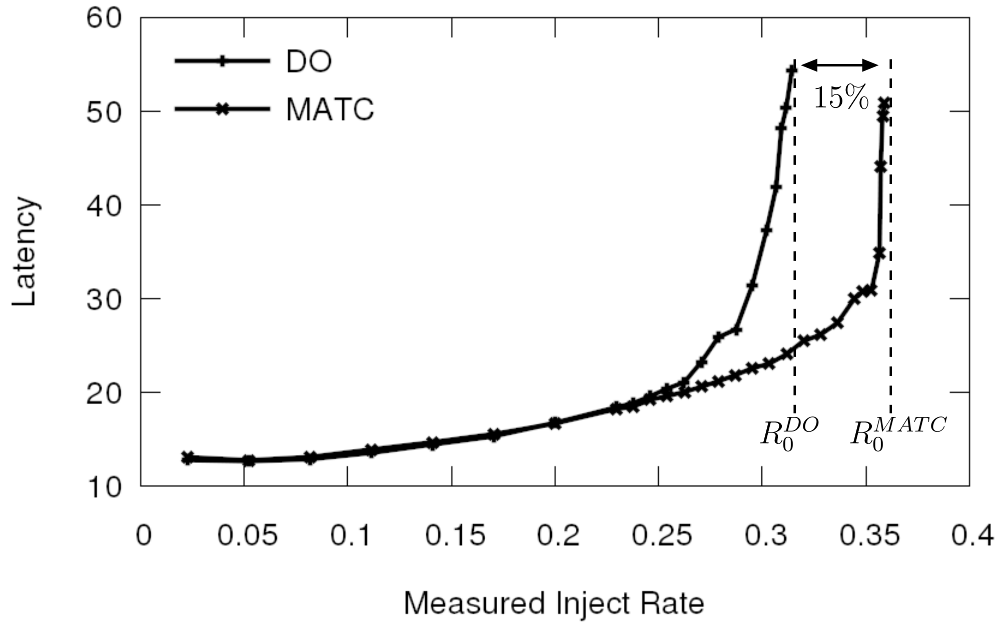


(a) $N(\mu = 1, \sigma = 0.07)$ (b) $N(\mu = 1, \sigma = 0.14)$ (c) $N(\mu = 1, \sigma = 0.21)$

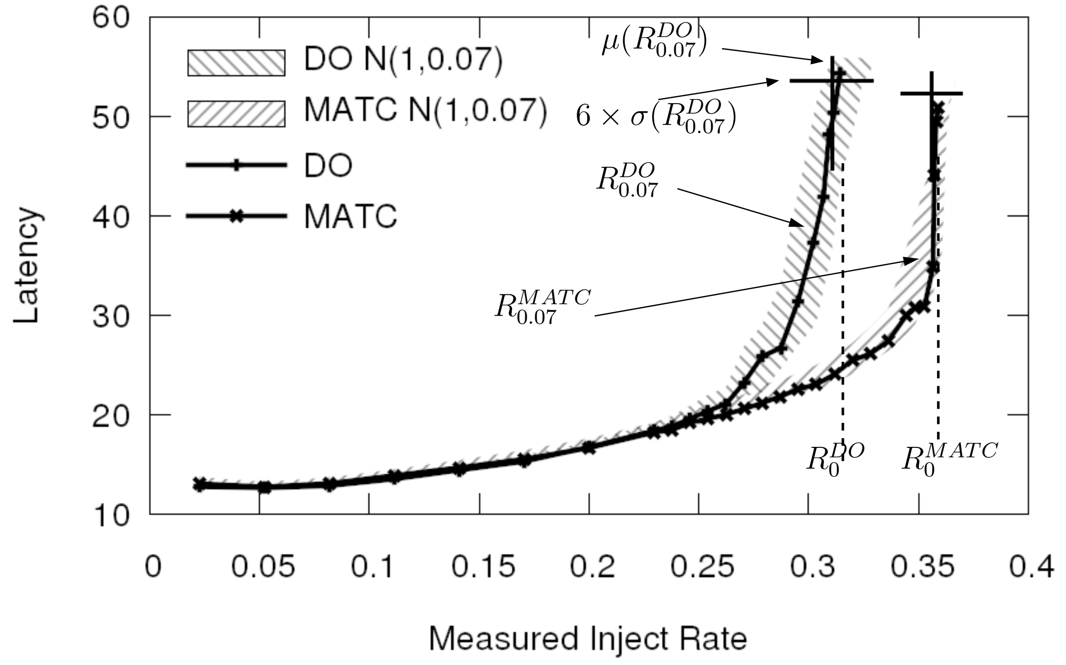
Figure 7.1: Performance model for normally distributed stochastic variations.

considered. Figure 7.2(a) shows the performance of Dimension Order (DO) and Minimal Adaptive Total Congestion (MATC) routers on a die with the nominal performance and the channel throughput $C_0 = 1$ flit per time unit. The **MATC** delivers approximately 15% saturation bandwidth advantage *vs* **DO**, *i.e.* $(R_0^{MATC}/R_0^{DO} = 1.15)$.

Figure 7.2(b) shows latency *vs* injection rate curves that demonstrate the impact of variations. The graph contains two curves for the *nominal* system with no performance variations, MATC and DO with $C_0 = 1$, and two curve "clouds" that represent the Monte Carlo experiment results over a space of channel performance $C_{0.07} = N(1, 0.07)$: MATC N(1,0.07) and DO N(1,0.07). These "clouds" illustrate the distribution of the latency *vs* bandwidth curves for the `PMEMD-64` application on a system with variations. Each curve cloud contains a cross, a vertical and a horizontal line. The vertical line marks the location of the statistical expected

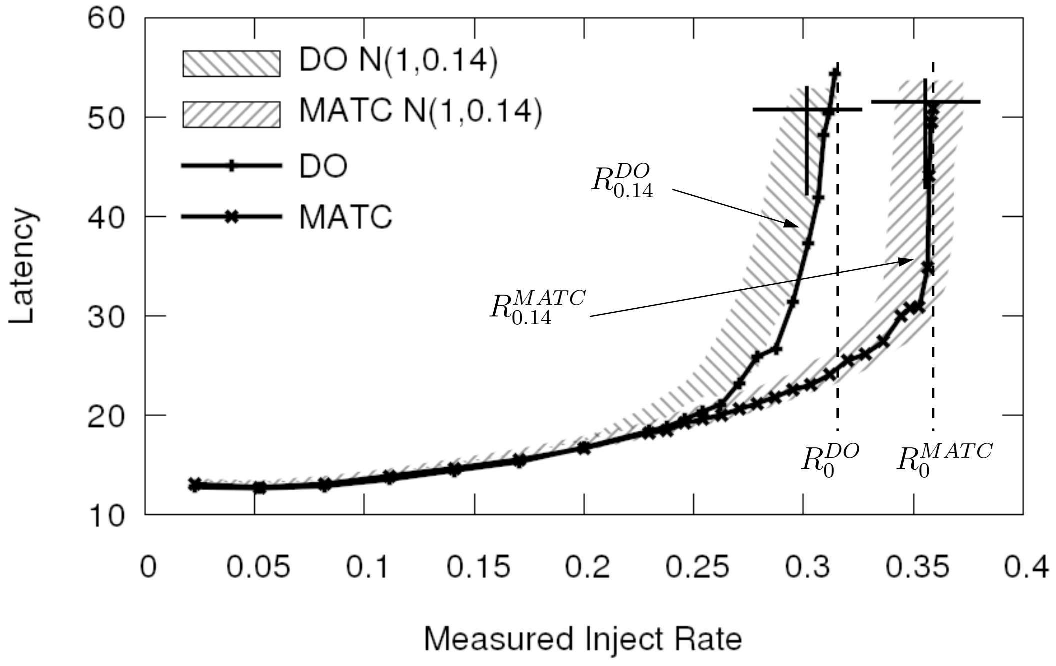


(a) No PVT variations

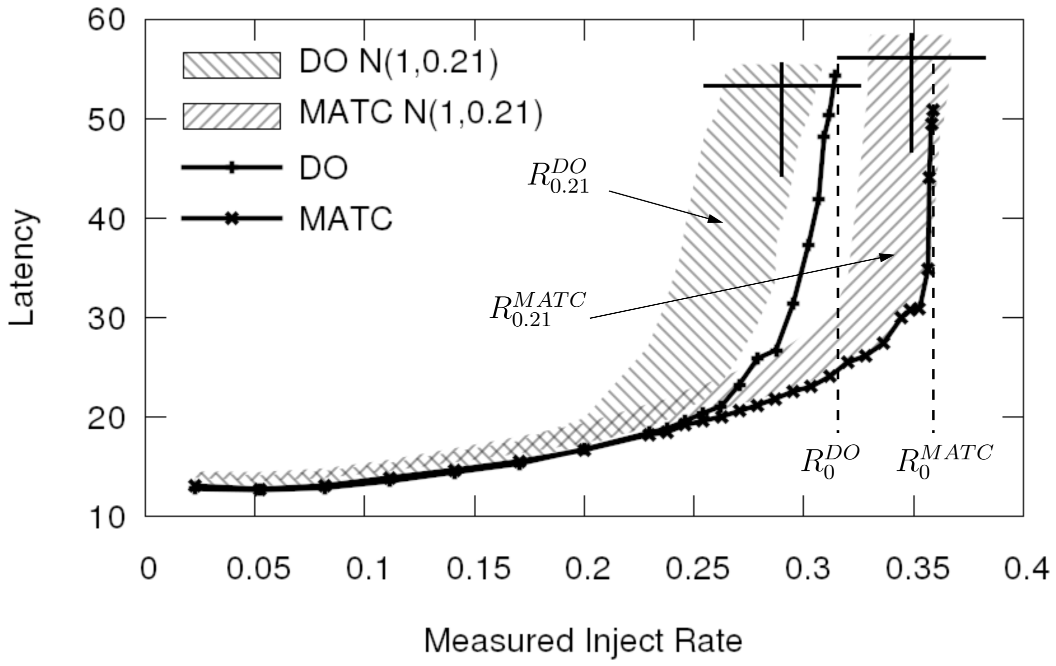


(b) $N(\mu = 1, \sigma = 0.07)$

Figure 7.2: MATC improves saturation bandwidth for `PMEMD-64` and compensates for PVT variations.



(a) $N(\mu = 1, \sigma = 0.14)$



(b) $N(\mu = 1, \sigma = 0.21)$

Figure 7.3: MATC improves saturation bandwidth for `PMEMD-64` and compensates for PVT variations.

value for the saturation injection rate $\mu(R^{\mathcal{R}}_{0.07})$. The horizontal line illustrates the $6 \times \sigma(R^{\mathcal{R}}_{0.07})$ span of the saturation injection rate.

Notice that the $6\times\sigma$ (six standard deviations) span reaches outside of the curve cloud. The reason is that the distribution of saturation injection rate does *not* have a normal or symmetric shape. It is a result of choosing a network path with the maximum throughput from a set of diverse paths in the topology, where the path throughput itself is constrained by its channel with the lowest throughput. Section 6.1 discusses of the PVT variation bounds, and Figure 2.4(b) illustrates the shape of the max and min functions applied to a set of stochastic variables. We use the standard deviation $\sigma(R^{\mathcal{R}}_{\sigma})$ simply as a measure of the distribution "width" to compare the routing algorithms and the applications.

With router throughput varying randomly, the network saturation bandwidth delivered by both the DO and MATC algorithms degrades relative to the respective nominal numbers. The expected values $\mu(R^{MATC}_{0.07})$ and $\mu(R^{DO}_{0.07})$ for the saturation injection rate are to the left of the nominal MATC and DO curves ($\mu(R^{\mathcal{R}}_{0.07}) < R^{\mathcal{R}}_{0}$). For `PMEMD-64` and router performance variation $C_{0.07}$, the network saturation bandwidth degrades by 1%.

However, the degradation increases sharply as the standard deviation of router throughput increases to $C_{0.14}$ on Figure 7.3(a) and to $C_{0.21}$ on Figure 7.3(b). Table 7.1 summarizes the impact of router performance variations on the saturation

For MATC vs DO, for PMEMD-64, the intrinsic improvement 15.4%.

Variations	Sat Bandwidth Degradation		Improv over Nominal DO	Improv over Realistic DO
C_σ	$\frac{\mu(R_\sigma^{DO})}{R_0^{DO}}$	$\frac{\mu(R_\sigma^{MATC})}{R_0^{MATC}}$	$\frac{\mu(R_\sigma^{MATC})}{R_0^{DO}}$	$\frac{\mu(R_\sigma^{MATC})}{\mu(R_\sigma^{DO})}$
$\sigma = 0.07$	0.99	0.99	1.13	1.15
$\sigma = 0.10$	0.98	0.99	1.14	1.16
$\sigma = 0.14$	0.96	0.99	1.13	1.18
$\sigma = 0.18$	0.94	0.97	1.11	1.18
$\sigma = 0.21$	0.92	0.97	1.11	1.20
$\sigma = 0.25$	0.91	0.97	1.11	1.22

Table 7.1: `PMEMD-64` Summary of saturation bandwidth degradation for DO and MATC.

bandwidth as $\mu(R_\sigma^{DO})/R_0^{DO}$ and $\mu(R_\sigma^{MATC})/R_0^{MATC}$, the ratios of the expected saturation rate for the case with performance variations to the saturation rate on the nominal system for the same routing algorithm. As the distribution C_σ widens, the network saturation injection rate $R_\sigma^{\mathcal{R}}$ decreases. As expected, the saturation bandwidth suffers significantly greater degradation for oblivious Dimension Order algorithm. The oblivious algorithm has a single fixed path between each source and destination pair, and does not take the advantage of network path diversity. Since the network throughput is only as large as the slowest router in the path, the DO algorithm is likely to encounter an outlier — an unduly slow router — that the algorithm cannot bypass. The adaptive algorithm MATC has the clear advantage in that it attempts to select a path with higher throughput from a collection of legal, minimal paths.

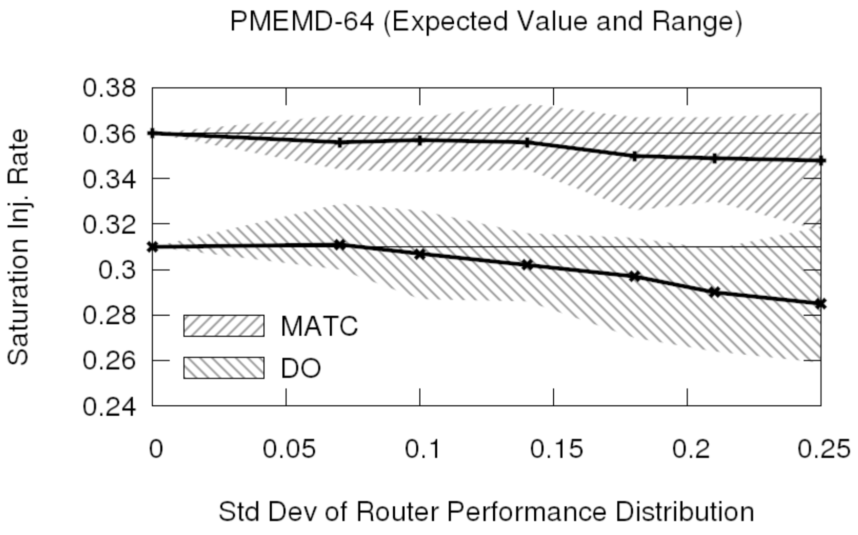


Figure 7.4: `PMEMD-64` Illustration of saturation bandwidth degradation for DO and MATC.

Figure 7.4 summarizes the impact of the router performance variations on the saturation injection rate of `PMEMD-64` application with DO and MATC routers. The graph shows the expected value $\mu(R^{\mathcal{R}}_{\sigma})$ with a line and the range of the saturation bandwidth distribution with a cloud. Two trends are clear. First, the expected saturation bandwidth $\mu(R^{\mathcal{R}}_{\sigma})$ degrades slower for the adaptive algorithm. Second, the adaptive algorithm reduces the standard deviation $\sigma(R^{\mathcal{R}}_{\sigma})$, which can translate into smaller design implementation guard-bands, less over-engineering and wasted resources. For this application, MATC has $R^{MATC}_0/R^{DO}_0 = 15.4\%$ "intrinsic" advantage on a nominal platform. Table 7.1 presents the net improvement of MATC on a realistic system with variations over the DO router on a

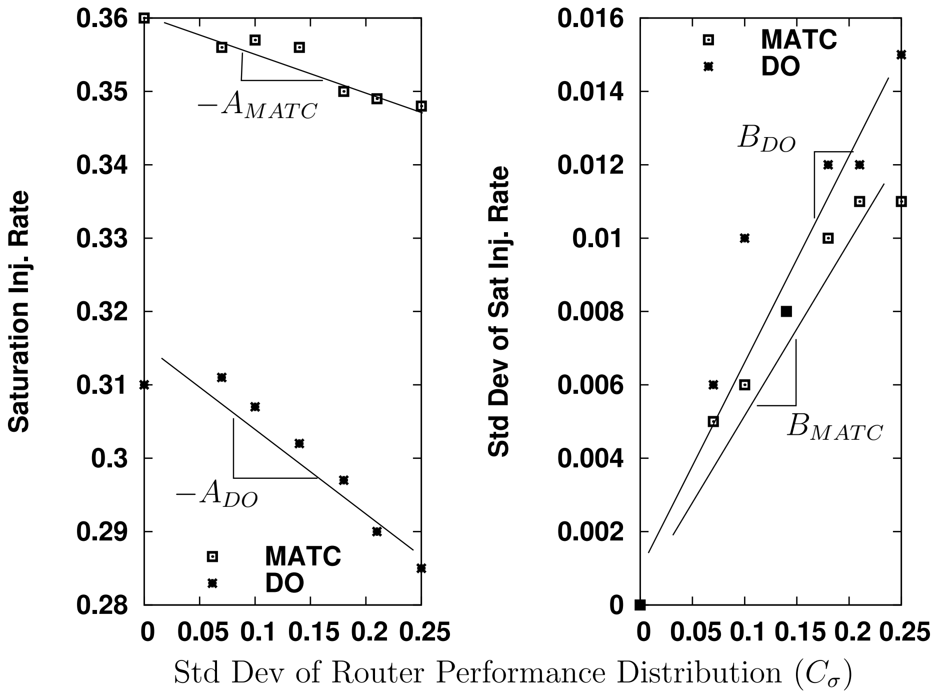


Figure 7.5: `PMEMD-64` Extracting parameters A and B from curves $\mu(R_\sigma^{\mathcal{R}})$ *vs* σ, and $\sigma(R_\sigma^{\mathcal{R}})$ *vs* σ.

nominal system: $\mu(R_\sigma^{MATC})/R_0^{DO}$. As the distribution of variations C_σ widens, the variations reduce this "intrinsic" advantage by a few percent.

How can we characterize the relationship between the magnitude of on-chip performance variations C_σ, and the expected value and the standard deviation of the saturation bandwidth distribution, $\mu(R_\sigma^{\mathcal{R}})$ and $\sigma(R_\sigma^{\mathcal{R}})$? Let us define a simple linear models with three parameters $R_0^{\mathcal{R}}$, $A_{\mathcal{R}}$ and $B_{\mathcal{R}}$:

$$\mu(R_\sigma^{\mathcal{R}}) = R_0^{\mathcal{R}} - A_{\mathcal{R}} \times \sigma \tag{7.6}$$

$$\sigma(R_\sigma^{\mathcal{R}}) = B_{\mathcal{R}} \times \sigma \tag{7.7}$$

Although, the actual relationship between these quantities is more complex than our model captures, in fact there are no closed forms, the equations above highlight the key general trends. The parameters, which can be extracted with a curve fit as shown on Figure 7.5, describe a routing algorithm's ability to contain and limit the impact of the PVT performance variations for a particular application communication pattern:

- Parameter $R_0^{\mathcal{R}}$ is the routing algorithm's saturation injection rate on a nominal NoC system. Ideally, this quantity should be close to the network topology bound. Any improvements in $R_0^{\mathcal{R}}$ come from the router's ability to compensate for communication load imbalance in an application task graph itself.

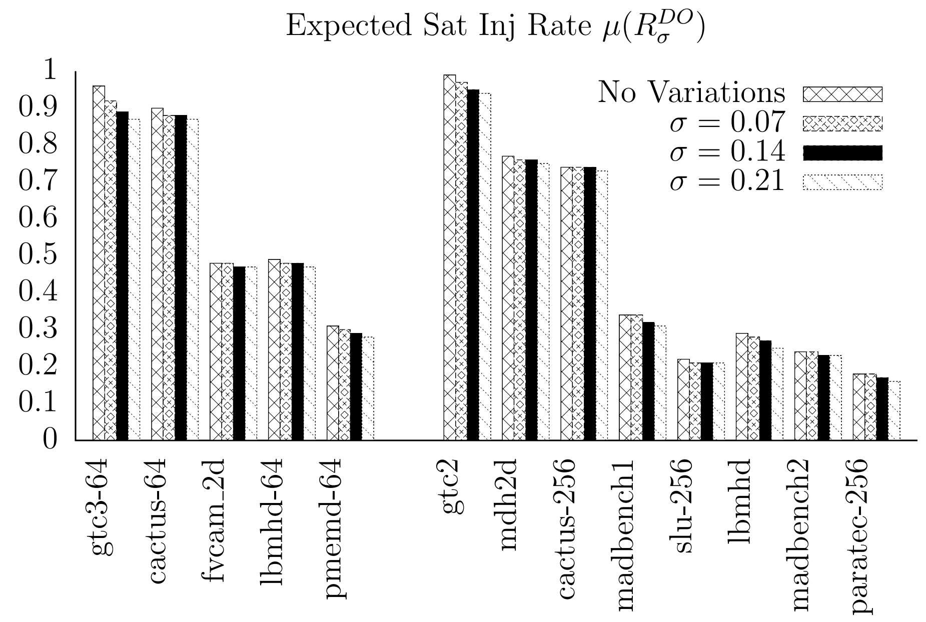
- Parameter $A_{\mathcal{R}}$ is the rate of degradation of the expected value of the saturation bandwidth $\mu(R_\sigma^{\mathcal{R}})$ relative to the standard deviation σ of the router performance distribution. This is one of the key metrics that characterizes the router's ability to minimize the impact of PVT variations. The more capable routing algorithms have a smaller value of $A_{\mathcal{R}}$. The ideal value of $A_{\mathcal{R}} = 0$ signifies that the algorithm *completely* compensates for performance variations. A negative value $A_{\mathcal{R}} < 0$ means that the router takes advantage of the performance heterogeneity to achieve even higher network saturation rate than possible in the nominal system.

- Parameter $B_{\mathcal{R}}$ is the rate of increase of the standard deviation of the saturation bandwidth distribution $\sigma(R_{\sigma}^{\mathcal{R}})$ relative to the standard deviation of the router performance distribution σ. This is another metric that describes the routing algorithm's ability to limit the impact of performance variations. Smaller values of $B_{\mathcal{R}}$ imply a better adaptive router. This parameter cannot be negative.

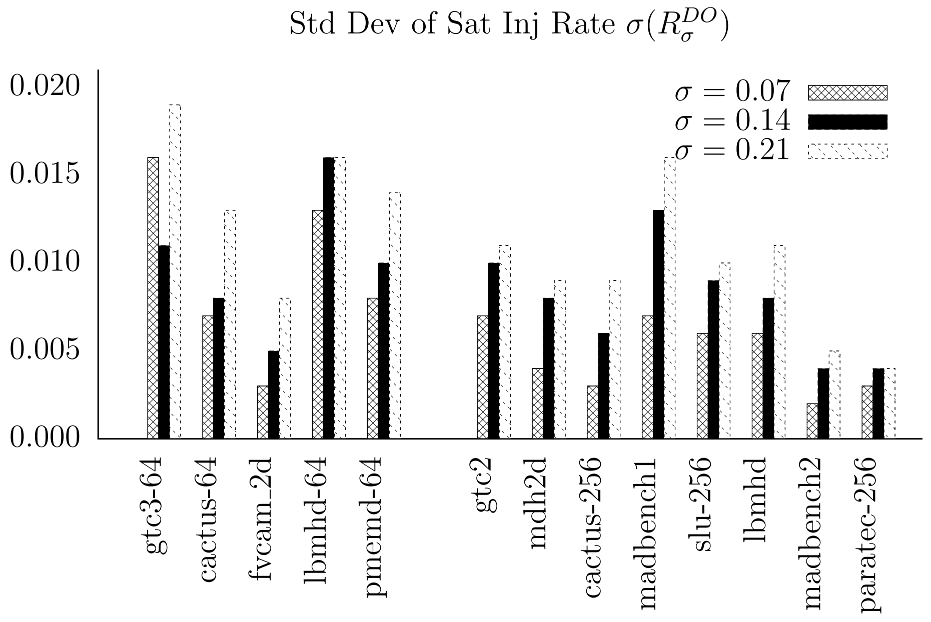
Figures 7.6 and 7.7 summarize the $\mu(R_{\sigma}^{\mathcal{R}})$ and $\sigma(R_{\sigma}^{\mathcal{R}})$ for the DO and MATC routing algorithms for all our applications. The applications are partitioned into 64 task group from `gtc3-64` to `pmemd-64`, and 256 task group from `gtc-2` to `paratec-256`, and sorted by the average communication distance within their respective groups. As previously discussed, the expected network injection saturation rates decrease with increasing communication distance. First, consider the expected saturation rate graphs on Figure 7.6(a) and 7.7(a). As the magnitude of performance variations C_σ increases, all applications experience a degradation in the saturation rate.

The standard deviations graphs on Figures 7.6(b) and 7.7(b) present a more complex picture. First, there is no consistent trend between the application's communication locality and the standard deviation of the saturation injection rate distribution, $\sigma(R_\sigma^{\mathcal{R}})$. The standard deviation additionally depends on the degree to which an application communication traffic is balanced and noisy local decisions of the distributed routing algorithm. Further investigation with more application graphs with diverse spatial communication characteristics are required to better understand the relationship between these factors, assuming there is a relationship. Second, in general as the magnitude C_σ of the router performance variations increases, so does the value of $\sigma(R_\sigma^{\mathcal{R}})$. This is expected, as the network saturation bandwidth is the *sum* of the bandwidths of individual channels across the topology bisection. An aggregation of random variables can reduce the standard deviation of the sum, but it does not eliminate it entirely. It is proportional to the standard deviation of the router performance.

To compare the ability of the DO and MATC routing algorithms to minimize the impact of PVT variations, we examine Figures 7.8 and 7.9. Figure 7.8 compares the network saturation injection rates for both algorithms on the nominal system, which highlights the "intrinsic" adaptive algorithm advantage.

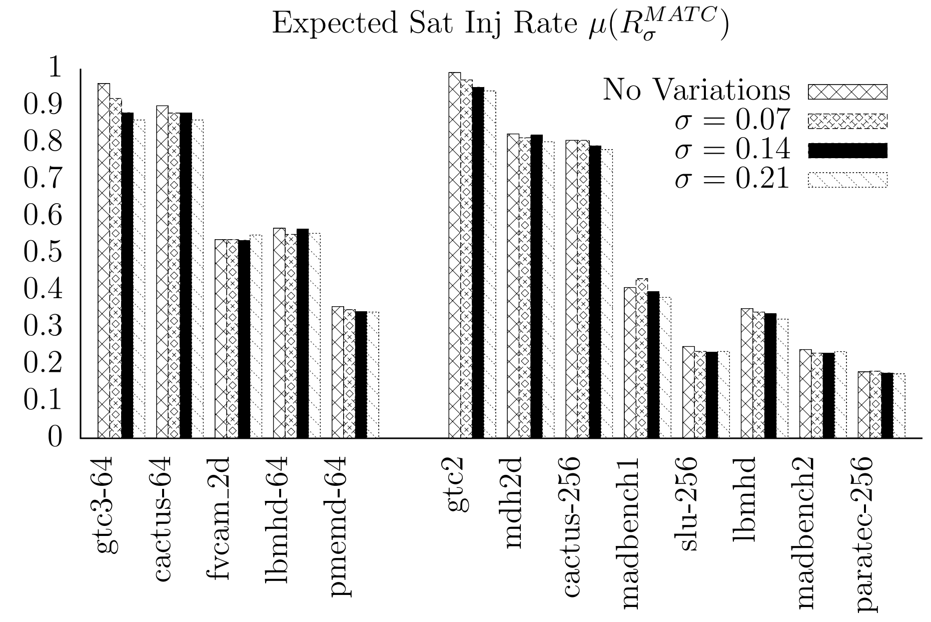


(a) Expected saturation injection rate $\mu(R_\sigma^{DO})$

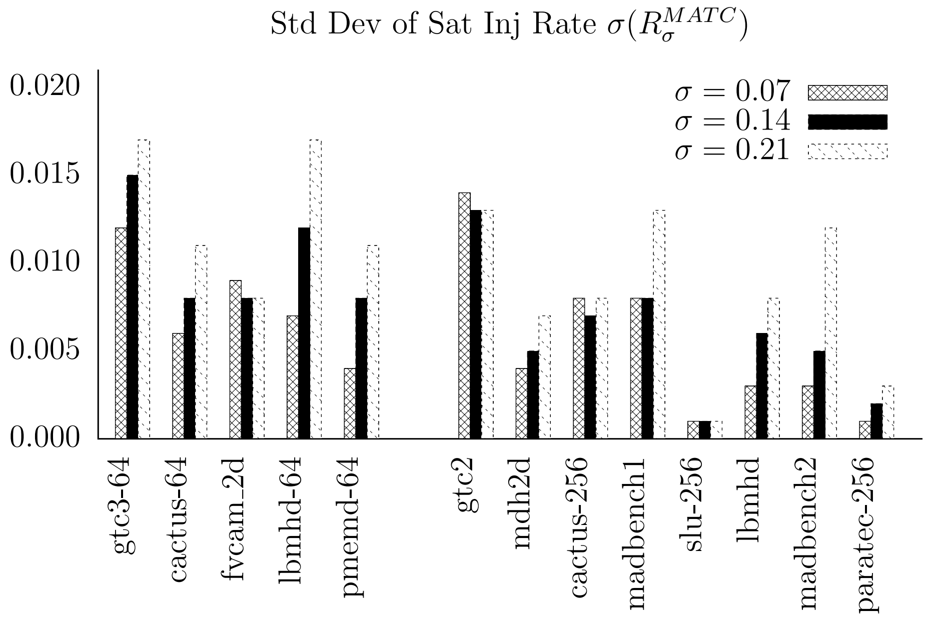


(b) Std Dev of saturation injection rate $\sigma(R_\sigma^{DO})$

Figure 7.6: Results summary for Dimension Order router.



(a) Expected saturation injection rate $\mu(R_{\sigma}^{MATC})$



(b) Std Dev of saturation injection rate $\sigma(R_{\sigma}^{MATC})$

Figure 7.7: Results summary for Minimal Adaptive router.

With exception of two applications `gtc3-64` and `gtc2` with very local, essentially nearest-neighbor, communication patterns, MATC demonstrates a higher saturation bandwidth than DO algorithm. The primary reason for the improvement is that these applications have an inherent communication load imbalance specific to their task-to-core mapping, and the adaptive algorithm routes the traffic and improves network channel utilization better than DO.

Figure 7.9 presents parameters A and B that compare the routers' ability to compensate for variations. With a few exceptions (such as `gtc3-64` and `cactus-64`), MATC has a lower value of A and minimizes the degradation of the network saturation bandwidth better than DO. The same trend applies for parameter B, whose value is generally lower for MATC algorithm. For some applications, such as `slu-256`, the parameter $B = 0$ with the adaptive router, meaning that the standard deviation for the saturation injection rate does not grow at all as router performance variations widen.

Application `madbench2` is an exception to this trend. The results obtained from distributed routing algorithms are ultimately specific to spatial communication patterns, and `madbench2` seems to contain a particular mix of local and global communication traffic that does not perform as well with a distributed adaptive router as it does with DO, in terms of parameters A and B. However, as

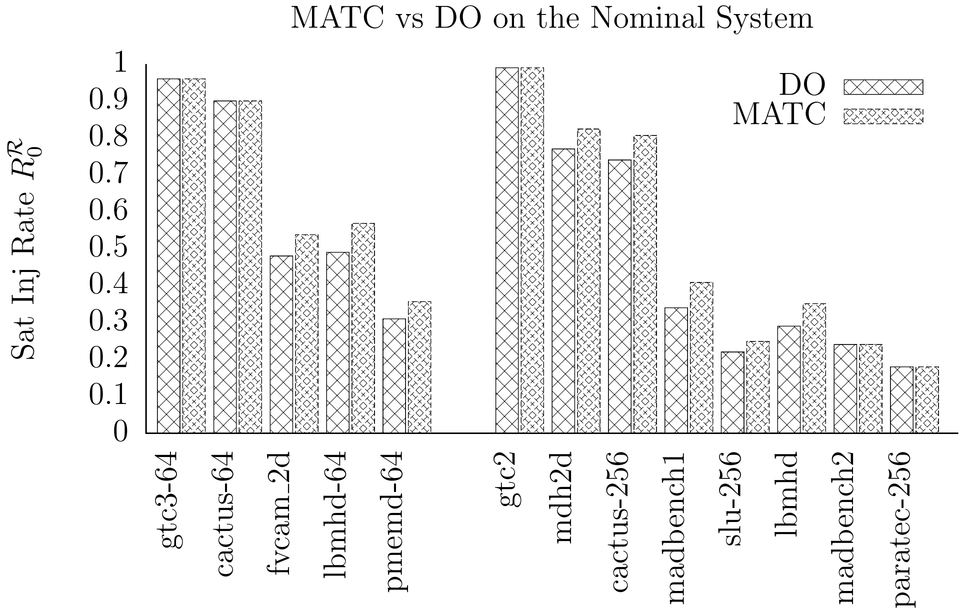


Figure 7.8: Saturation bandwidth for MATC and DO on the nominal NoC.

Figure 7.8 shows, MATC still outperforms DO for `madbench2` in absolute terms of the saturation injection rate.

7.2.2 Systematic Variations

In addition to considering purely stochastic router performance variations, let us investigate a systematic effect. As discussed in Section 2.1, the exact systematic on-chip performance profile depends largely on the semiconductor manufacturing process as well as the die layout and architecture. For our experiment, we consider a linear gradient variation profile illustrated on Figure 7.10, where the router

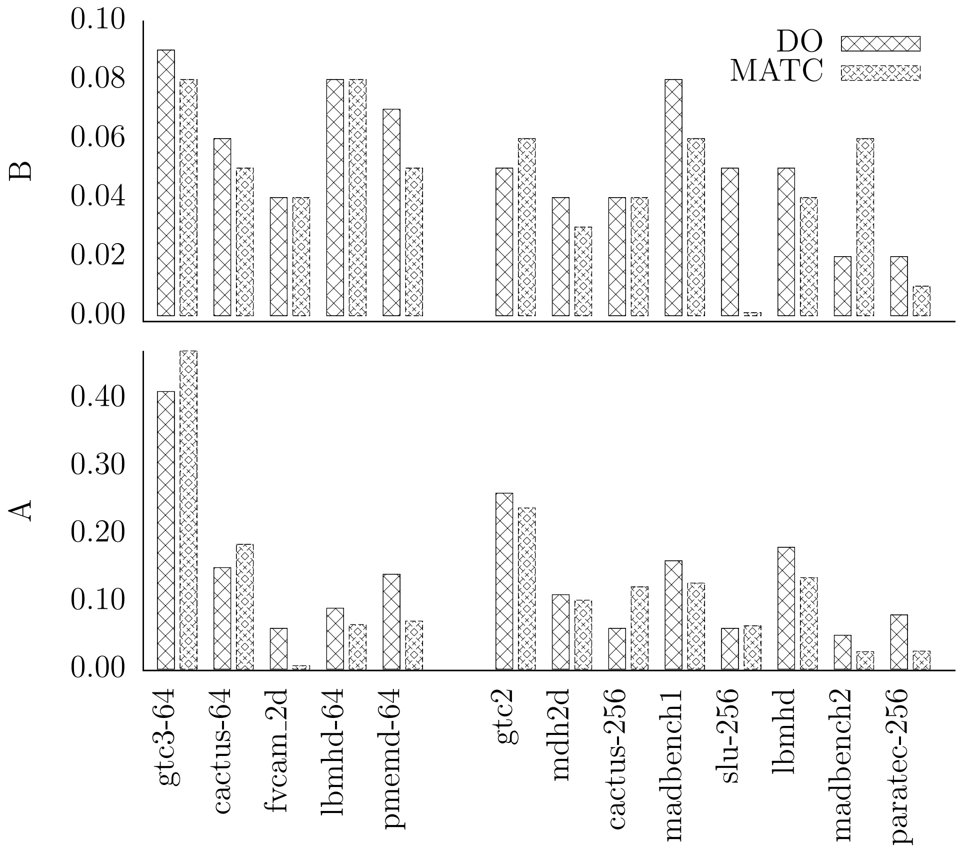


Figure 7.9: Routing algorithm variation sensitivity parameters A and B.

throughput increases with its on-die position from left to right. The gradient is designated as $SP(m, n)$, where m and n represent minimum–maximum throughput range. The router performance C can be described by the following mapping:

$$C : (x, y) \mapsto K\left(\frac{m-n}{w}x + m\right) \tag{7.8}$$

where the tuple (x, y) is router position, w is the die size in X dimension, and K is a scaling coefficient. Unlike stochastic performance variations that have an unpredictable die performance profile, this particular performance gradient and its orientation favor certain application graphs and their task-to-core assignments. Some task-to-task communication flows may pass through the faster part of the chip, while others may be disadvantaged. The goal of an adaptive routing algorithm is to minimize any disadvantages and to reduce NoC load imbalance independent of the variation type or source.

Figure 7.11 shows the simulation results for DO and MATC algorithms on the systems with nominal and systematic performance gradients for `PMEMD-64` application. First, consider the results of the Dimension Order routing algorithm. The saturation bandwidth numbers for DO SP are around the nominal value. `PMEMD-64` takes advantage of this particular spatial performance gradient and its orientation. DO SP(0.9,1.1) has a higher saturation bandwidth than the nominal

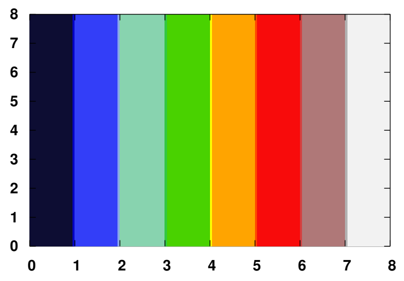


Figure 7.10: An example of a Systematic Performance Gradient.

DO. DO SP(0.8,1.2) has about the same as the nominal system. Only with a very steep gradient, DO SP(0.7,1.3) has the lower performance than the nominal system, which is the typical and expected impact of the PVT variations.

MATC compensates for the performance gradient completely. All MATC SP curves are essentially the same and *outperform* the nominal system. In this case, the adaptive algorithm actually takes advantage of heterogeneity in the NoC router performance and finds packet routes to deliver greater saturation bandwidth than the system without variations would allow. Note, however, that the results of the adaptive algorithm are very specific to the task-to-node assignment, *i.e.* the orientation of the task graph on the die.

The relationship between the saturation injection rate and the gradient of this systematic on-chip performance variation can be characterized with a simple

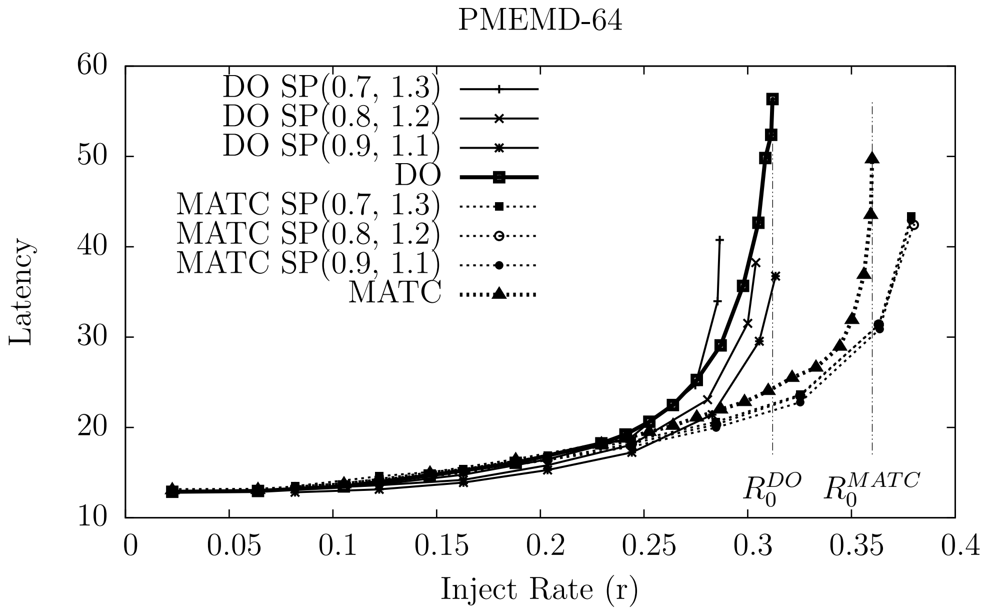


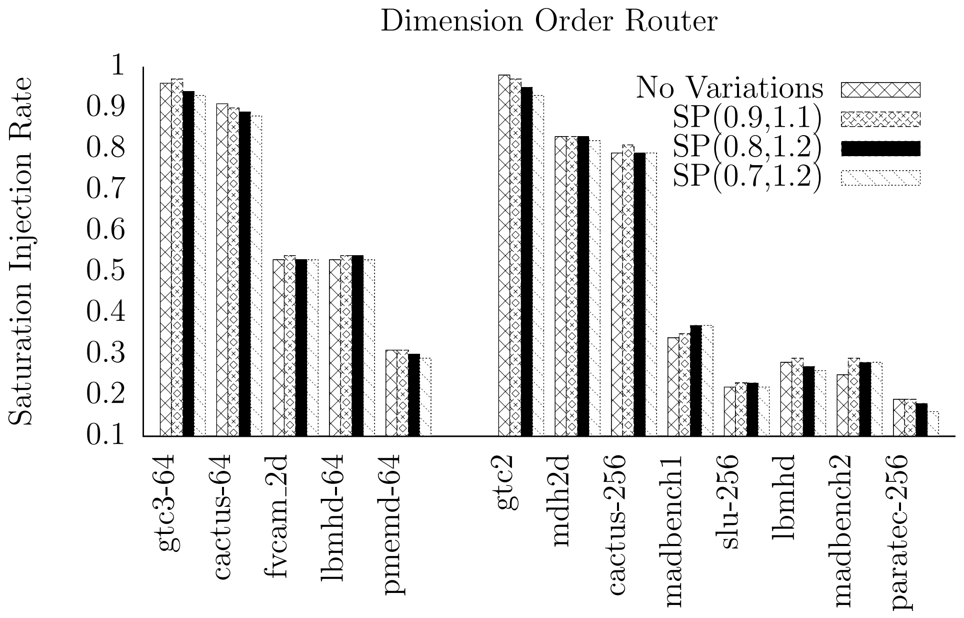
Figure 7.11: MATC completely compensates the bandwidth degradation.

model with two parameters $R_0^{\mathcal{R}}$ and $A_{\mathcal{R}}$:

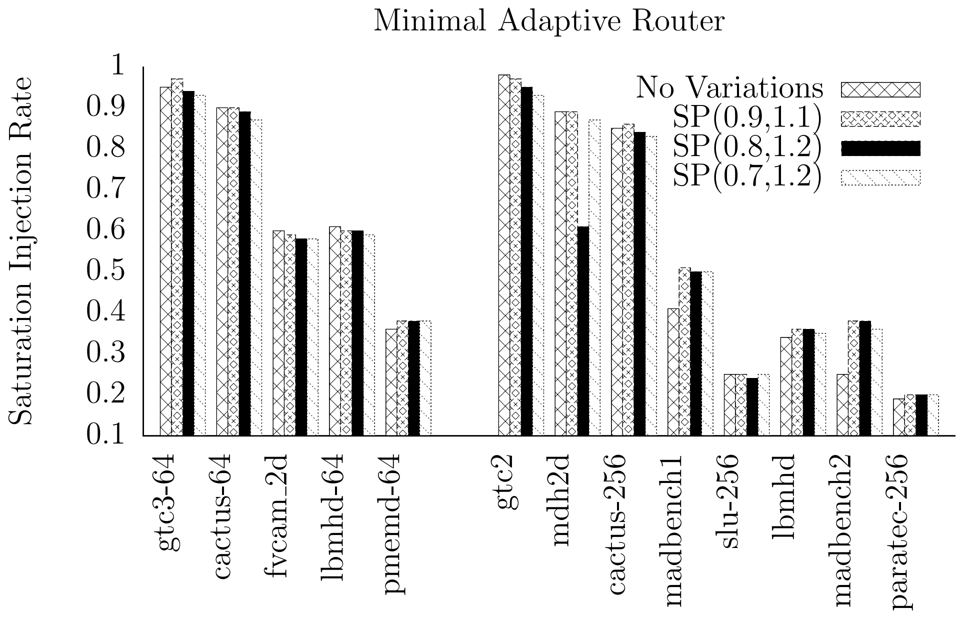
$$R_{m,n}^{\mathcal{R}} = R_0^{\mathcal{R}} - A_{\mathcal{R}} \times |m - n| \tag{7.9}$$

Similar to the discussion in the previous section, parameter $R_0^{\mathcal{R}}$ represents the network saturation injection rate for the routing algorithm $\mathcal{R}$. Parameter $A_{\mathcal{R}}$ characterizes the algorithm's ability to contain and minimize the impact of on-die performance variations.

Figures 7.12(a) and 7.12(b) summarize the network performance results for the DO and MATC routing algorithms for our applications. The same general



(a) Dimension Order



(b) Minimal Adaptive

Figure 7.12: Results summary for a systematic performance gradient.

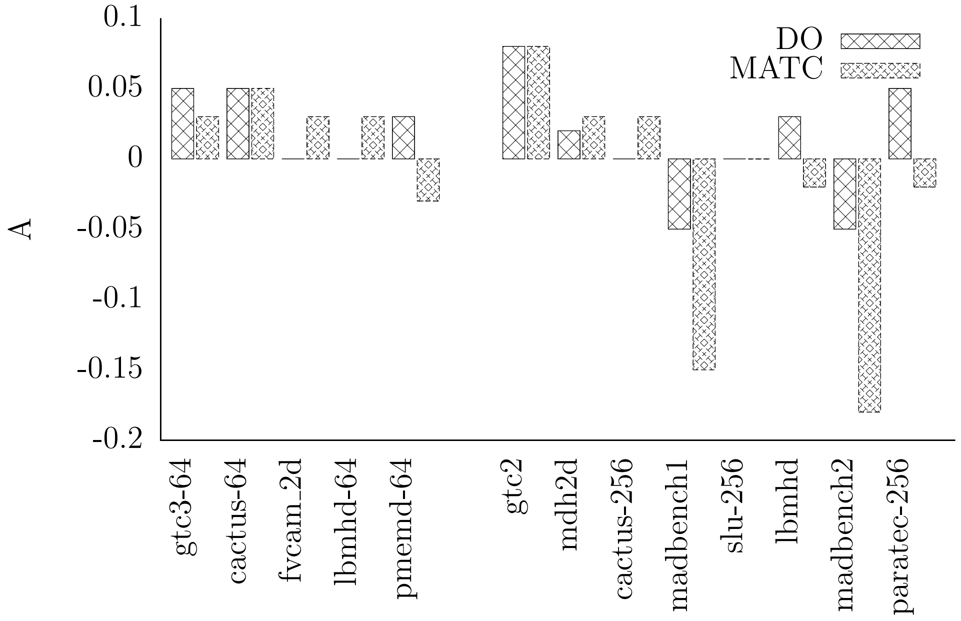


Figure 7.13: Parameter A is the slope of the degradation of the saturation injection rate on a NoC with the performance gradient.

trends that were observed for the stochastic PVT variations apply here. Typically, as the gradient $SP(m, n)$ of the on-chip grows steeper, the network saturation rate decreases. Several notable and significant exceptions to this trend include `madbench1`, `madbench2`, `lbmhd` and `paratec-256` applications with the adaptive MATC routing algorithm. For these cases, router performance heterogeneity and more specifically the $SP(m, n)$ gradient are a good match for the spatial communication patterns of these applications, which results in the saturation bandwidth improvement.

Figure 7.13 summarizes the simulation results on the system with the systematic gradient. Parameter A describes the degree to which a routing algorithm can minimize the impact of variations. The graph shows no consistent trend for the smaller 64 task applications, but a consistent pattern of $A_{MATC} < A_{DO}$ for the large applications, which indicates that MATC minimizes the saturation bandwidth degradation better than DO. Some applications, such as `madbench1` and `madbench2`, even have a negative value of A, in cases where the NoC with performance variations outperforms the nominal system. This result of a fortunate match between the application heterogeneous communication resource requirements and those of the underlying network further underscores the ability to adaptive routing to exploit heterogeneity.

7.2.3 Stochastic and Systematic Results Summary

Figure 7.14 contains the grand summary of the results. It highlights the improvement in the saturation injection rates with MATC *vs* DO as the ratio $\mu(R_{\sigma}^{MATC})/\mu(R_{\sigma}^{DO})$. The higher ratio represents the greater improvement in the network performance, the higher saturation bandwidth and thus the lower average packet latency over a wider network bandwidth operating region. The performance advantage of adaptive routing is a result of two factors:

1. The adaptive algorithm compensates for application specific load imbalance on the nominal system (Figure 7.8). In other words, $R_0^{MATC} > R_0^{DO}$.

2. The adaptive algorithm further reduces the degradation of the network saturation bandwidth caused by the variations and minimizes their negative impact on the system. $A_{MATC} < A_{DO}$ and $B_{MATC} < B_{DO}$.

Let us first consider the summary for the *stochastic* performance variations on Figure 7.14(a) and identify two trends. First, as the average communication distance increases, the applications show greater saturation bandwidth improvement with MATC than with DO. Consider `gtc3-64`, `cactus-64` and `gtc2` applications whose average communication distance does not exceed 2 router hops (Table 8.2). These application simply do not offer enough path diversity to an adaptive routing algorithm and show no appreciable improvement in network saturation bandwidth. Task-to-core mapping quality has the dominant impact on the network performance of the applications with local communication patters (Section 7.5). In contrast, the applications with more global communication pattern demonstrate a significant 7–25% improvement in the network saturation bandwidth with MATC *vs* DO. Applications `madbench2` and `paratec-256` deviate from this trend. Although, their spatial communication pattern is global, a careful analysis reveals that these applications have a communication pattern that closely resembles the uniform random, *i.e.* the communication load is basically balanced across NoC resources. For these applications, MATC and DO deliver approximately equivalent performance because the adaptive algorithm cannot improve the network utilization further with an already balanced communication load.

The second trend can be identified by comparing the MATC/DO ratios within each application. In general, as performance variations grow from $\sigma = 0.07$ to $\sigma = 0.21$, so does the ratio $\mu(R_\sigma^{MATC})/\mu(R_\sigma^{DO})$. Particularly good examples include `fvcam_2d`, `lbmhd-64`, `pmemd-64`, `madbench1` and `lbmhd`, which show clear improvements. They indicate that the saturation bandwidth degrades slower with MATC algorithm as compared to the oblivious DO router.

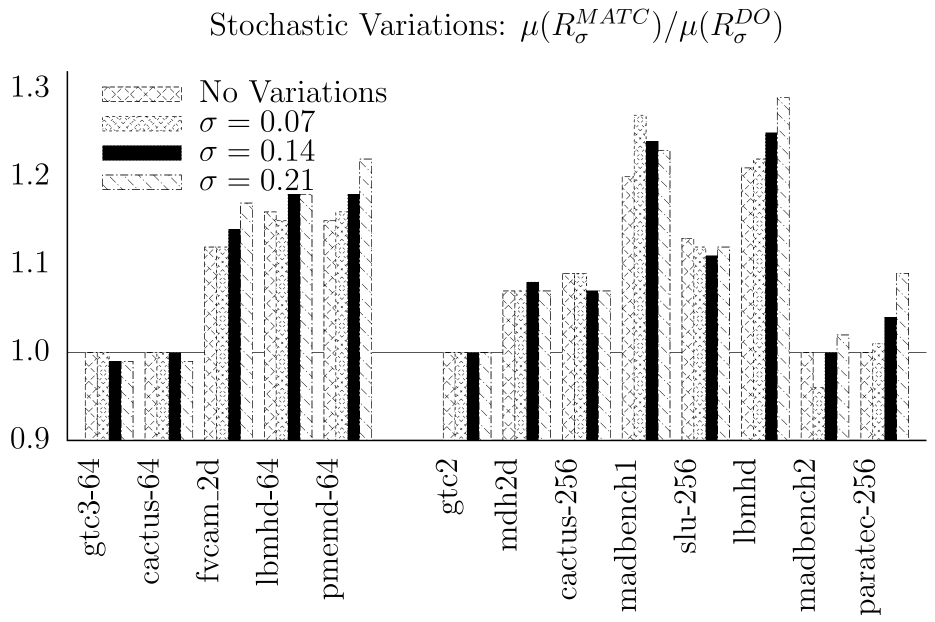
Application `paratec-256` highlights clearly the ability of an adaptive algorithm to tolerate PVT variations. From the point of view of a spatial communication pattern, `paratec-256` is very similar to the uniform random traffic, which is naturally load balanced. The performance variations create NoC resource utilization imbalance, which MATC restores and obtains a greater performance improvement than DO.

Figure 7.14(b) summarizes the simulation results for a NoC with our simple *systematic* gradient, which demonstrate very similar trends to the stochastic on-chip performance profile. Particularly notable are the saturation bandwidth improvements on applications with non-local communication traffic. The systematic gradient $SP(m, n)$ is simple and regular, and a relatively easy case for any adaptive algorithm to manage. This explains the larger net saturation bandwidth improvements of 5–35% than those on NoC systems with stochastic performance variations.

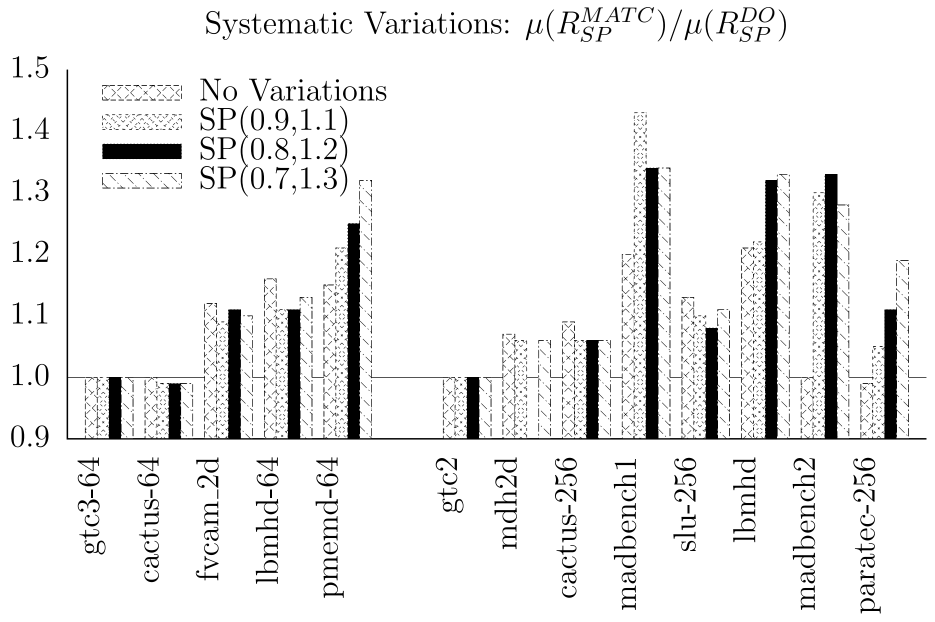
The results highlight the key feature of our proposed high level approach. Adaptive NoC routing can effectively compensate for a range of PVT variations independent of their nature and their specific sources, as long as the architecture exposes the variations as component performance. For routing, exposing PVT variations as flow control and router congestion enables a simple and comprehensive approach to mitigate the degrading impact of variations.

7.3 Reducing Design Guard-bands

Let us define a simple model the relates design guard-bands to the improvements in the saturation bandwidth that MATC can deliver over the Dimension Order routing algorithm. Although the following discussion focuses on stochastic variations, a similar model applies to a systematic profile as well. C_σ describes the router performance in an implementation of the multi-core NoC architecture, *i.e.* the router throughput is described as a random variable with Normal distribution $N(\mu, \sigma)$. μ represents the target (nominal) router throughput chosen by system



(a) Stochastic Variations



(b) Systematic Variations

Figure 7.14: MATC *vs* DO improvements in saturation injection rate.

implementers[1]. σ describes the "width" of the router throughput distribution in a system affected by PVT variations. Network saturation injection rate $R_\sigma^{\mathcal{R}}$ is the key metric that defines the upper bound on the operating region where NoC system functions efficiently. Equations 7.6 and 7.7 modeled the relationship between C_σ and $R_\sigma^{\mathcal{R}}$, and are repeated here for clarity:

$$\begin{aligned} \mu(R_\sigma^{\mathcal{R}}) &= R_0^{\mathcal{R}} - A_{\mathcal{R}} \times \sigma & (7.10) \\ \sigma(R_\sigma^{\mathcal{R}}) &= B_{\mathcal{R}} \times \sigma & (7.11) \end{aligned}$$

The nominal network saturation injection rate $R_0^{\mathcal{R}}$ is a function of the target router throughput μ: $R_0^{\mathcal{R}} = f(\mu)$. This function is linear $f(k\mu) = kf(\mu)$, because the network saturation bandwidth is proportional to the sum of individual channel capacities across the topology bisection. The linearity enables a simple definition of the design guard-band g:

$$(1+g) \times R_0^{\mathcal{R}} = f\left((1+g) \times \mu\right) \qquad (7.12)$$

The guard-band g represents an additional performance margin that must be built into the design in order to meet the network saturation bandwidth specification S. If $6 \times \sigma$ parametric yield is desired, then S constrains the network performance

[1]In the previous sections, we set $\mu = 1$ flit per time unit for simplicity.

as following:

$$\mu(R_\sigma^{\mathcal{R}}) - 3\sigma(R_\sigma^{\mathcal{R}}) \geq S \tag{7.13}$$

Substituting the definitions for $\mu(R_\sigma^{\mathcal{R}})$ and $\sigma(R_\sigma^{\mathcal{R}})$, we obtain:

$$(1+g) \times R_0^{\mathcal{R}} - A_{\mathcal{R}} \times \sigma - 3B_{\mathcal{R}} \times \sigma \geq S \tag{7.14}$$

$$g \geq \frac{S - R_0^{\mathcal{R}} + (A_{\mathcal{R}} + 3B_{\mathcal{R}}) \times \sigma}{R_0^{\mathcal{R}}} \tag{7.15}$$

This simple definition for a design guard-band allows us to evaluate the results in the previous sections from a different angle. Unlike the results in Figure 7.14 that highlighted improvement in the *expected* saturation bandwidth between the two routing algorithms, the guard-band comparison also factors in the standard deviation of the network saturation bandwidth distribution. Figure 7.15 presents the guard-band g for MATC with variations $C_{0.21}$, where the specification S was chosen such that the guard-band g for DO would be 0: $S = R_0^{DO} - (A_{DO} + 3B_{DO}) \times \sigma$. The negative guard-band values of g show that at their nominal throughput of $\mu = 1$ flit per time unit, the MATC routers are over-engineered for the specification point S, and thus the nominal router throughput μ can be

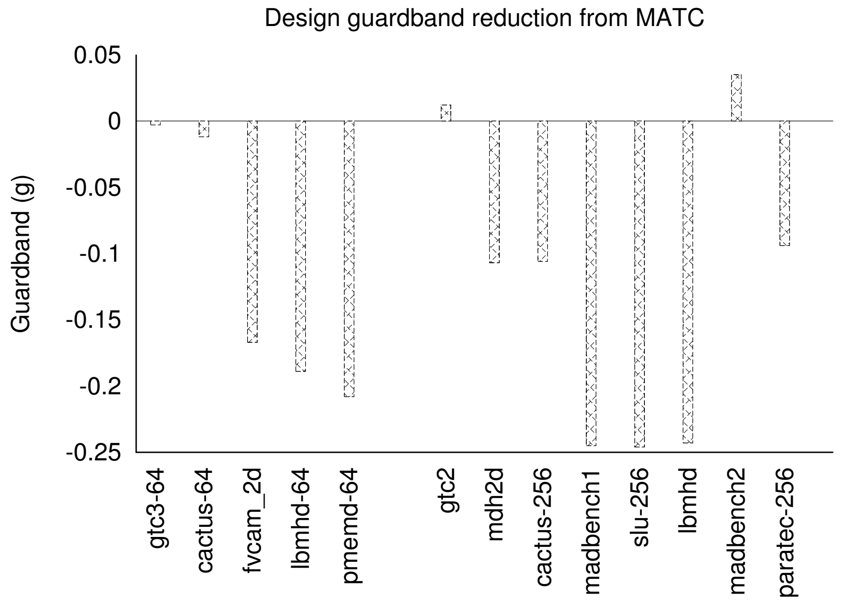


Figure 7.15: Guard-band reduction with MATC adaptive routing with stochastic variations $C_{0.21}$.

reduced. It is not surprising that the qualitative trends in Figure 7.15 resemble those in the previous sections.

Figure 7.16 illustrates the general strategy to convert the performance gains from adaptive routing into smaller design guard-bands. Without adaptive routing, a NoC would be implemented in a particular technology to deliver the router performance profile C_σ. MATC reduces the degradation and the standard deviation of the saturation bandwidth and enables a designer to select a design point $\overline{C_\sigma}$ with lower performance target. The new design point $\overline{C_\sigma}$ corresponds to smaller area and power dedicated to deliver the target performance.

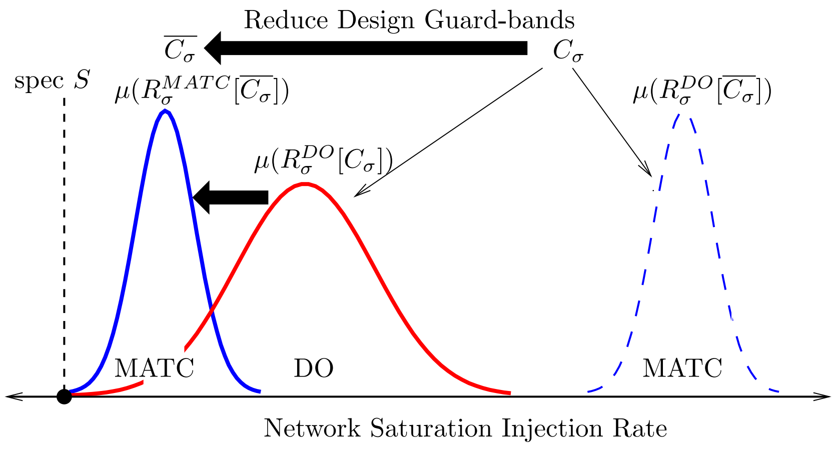

Reduce Design Guard-bands
$\overline{C_\sigma}$
C_σ
spec S
$\mu(R_\sigma^{MATC}[\overline{C_\sigma}])$
$\mu(R_\sigma^{DO}[C_\sigma])$
$\mu(R_\sigma^{DO}[\overline{C_\sigma}])$
MATC
DO
MATC
Network Saturation Injection Rate

Figure 7.16: From improvements in saturation bandwidth to reduced design margins.

Adaptive routing gives implementers a performance and power advantage, but the expected gains depend on the application and its communication properties. The application domain can help to determine the best way to realize the gains from adaptive routing. Assume that NoC system's application load can be characterized in advance and generally comprises applications with similar communication patterns in terms of locality and load balance. A designer can set the target implementation point $(1+g)\times\mu$ to lower the area and power consumption. This design time decision combines with a dynamic adaptive routing algorithm to reduce the resources required to meet the hard performance specification S.

In practice, it is unlikely that the application domain is sufficiently narrow in terms of its members' communication properties, requiring a combination of design and run time techniques. At design time, it is possible to reduce the guard-bands just enough to accommodate the majority of the application domain. Then at run-time, DVS/DFS techniques can extend the operating range of routers to accommodate all applications that execute on a platform.

This combination results in the benefits of both worlds. A designer reduces power and area by setting the nominal design point in the middle of the operating range that would accommodate all applications. The run-time systems selects operating points for each core/router based on its actual load. This can be accomplished with an architecture that exposes PVT induced variations and operating regime controls to the run-time system and allows a routing algorithm to treat these complex effects as ordinary network congestion. The approach accommodates a variety of static variation effects, such as process variations and faults, dynamic effects induced by temperature and voltage fluctuations, and the application specific load imbalance and device aging.

7.4 Benefits of Heterogeneous Architectures

This work demonstrates the advantages of adaptive *vs* oblivious routing on heterogeneous multi-core architectures, where each core/router tile operates in its own independent voltage and clock domain. These Voltage Frequency Islands

(VFI) allow each router to run at the throughput and latency dictated by its own local PVT variation corner. The alternative, of course, would be to force all tiles to operate at the same chip-wide *worst-case* performance point, or perhaps to select a coarser VFI granularity with core clusters.

Our preliminary investigation illustrates the advantages of the core-level Voltage Frequency Island architectures on Figure 7.17. The graph shows the network performance for `PMEMD-64` application on the network affected by stochastic performance variations distributed as $N(1, 0.21)$. These results were obtained with Minimal Adaptive (MA) and Dimension Order (DO) routers. There are two sets of curves: (1) "Homogeneous" multi-core system without VFIs — all core/router tiles operate at the chip-wide worst case (WC) regime; and (2) "Heterogeneous" system where every core has its own VFI, which is th focus of this work. For this application, $R^{MATC}_{VFI}/R^{MATC}_{WC} = 1.42$ and $R^{DO}_{VFI}/R^{DO}_{WC} = 1.36$. In other words, the performance heterogeneity with VFIs provides approximately 40% saturation bandwidth improvement independent of the routing algorithm.

In the "Homogeneous" architecture all tiles run at the same, worst case operating point. The PVT variations are not exposed to the routers, and the significant network performance degradation is the result of accommodating all tiles, including the unduly slow outliers. The graph identifies three important performance improvement regions:

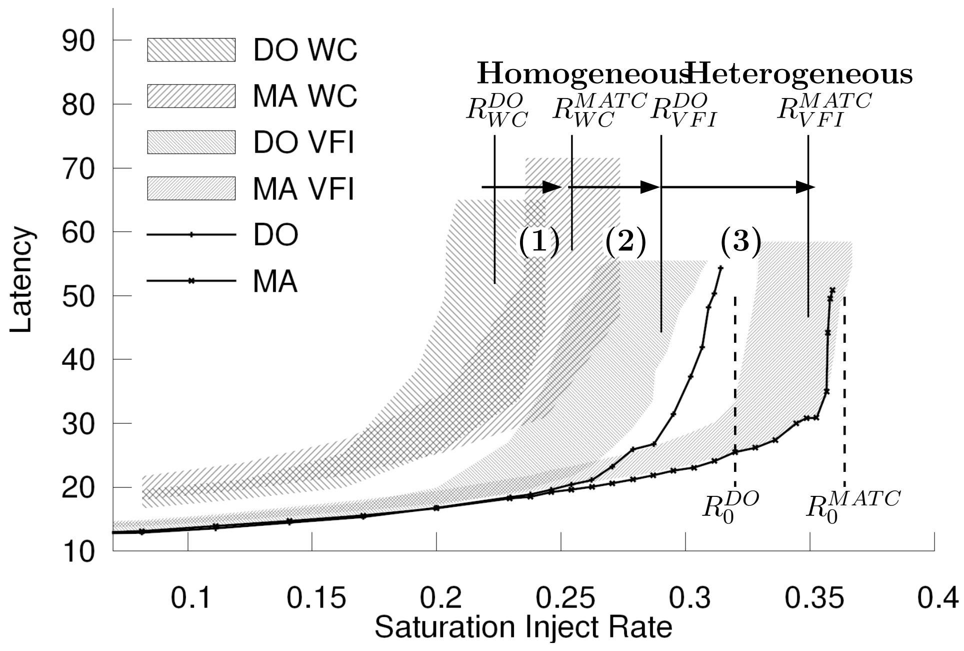


Figure 7.17: Advantages of heterogeneous *vs* homogeneous architectures: 40% gains in saturation bandwidth, and 50% in unloaded latency.

- Region (1) shows the saturation bandwidth improvement from adaptive routing that compensates for the application communication load imbalance.

- Region (2) highlights the potential performance difference between the two extremes: homogeneous and heterogeneous architectures with a core per VFI. In practice, there is a continuum between these extremes that contains architectures where each Voltage Frequency Island is a cluster of cores. Depending on the core size and capabilities, and whether or not a core includes caches, clustering of the adjacent cores into a VFI may be appropriate to create an efficient implementation. A number of other variants exist, such a cluster of cores that share a cache block or a network-on-chip router (*e.g.* concentrated network topologies [9]), which have additional area and power advantages.

- Region (3) highlights the adaptive routing advantages that are the focus of this work. This region is wider than region (1) because an adaptive algorithm compensates for communication load imbalance due to PVT variations as well as the imbalance inherent in the application communication topology and task-to-core mapping (Section 7.2.3).

One of the key questions to answer in the future work would be to quantify the gains from heterogeneous multi-core architectures, *i.e* the region (2). To properly understand this space, one must obtain the results from a set of implementations with concrete tile operating ranges and vary the VFI granularity (the number of cores per VFI). This would highlight the trade-off between the VFI implementation overhead and the potential gains from adaptive routing and task mapping, and would enable architects to identify the most power/performance efficient system implementation.

7.5 Task-to-Core Mapping Impact

The network simulation results presented in Section 7.2 relied on task-to-core mapping performed by the simulation annealing algorithm in the VPR [16]. The algorithm attempts to minimize the average communication distance by minimizing the bounding box size that encompasses every a pair of communicating tasks on a 2D mesh. This Minimal Bounding Box algorithm (Min BB) maps or "places" the communicating tasks on adjacent cores and preserves spatial locality in the application communication pattern. Table 8.2 presents two Rent's growth parameters for each application task graph: Partition(p) and Placed(p). Partition(p) was obtained by recursive task graph bisection without any task-to-core map-

Name	Size	Avg Comm Dist D_{avg}		Rent p	
		Min BB	Random	Min BB	Random
gtc3-64	64	1.09	5.44	0.26	0.63
cactus-64	64	1.73	5.28	0.30	0.67
fvcam_2d	64	2.61	5.32	0.47	0.64
lbmhd-64	64	3.36	5.52	0.51	0.69
pmemd-64	64	5.24	5.42	0.66	0.66
gtc2	256	1.12	11.02	0.26	0.68
mdh2d	256	2.02	10.84	0.31	0.70
cactus-256	256	2.12	10.40	0.32	0.66
madbench1	256	3.31	10.63	0.45	0.67
slu-256	256	5.72	10.86	0.57	0.69
lbmhd	256	5.81	10.60	0.53	0.67
madbench2	256	5.83	10.42	0.49	0.66
paratec-256	256	10.66	10.67	0.68	0.68

Table 7.2: Random placement destroys communication locality.

ping. Placed(p) was obtained by first mapping the communication graph using VPR and then extracting the parameters by bisecting the 2D network topology. Partition(p) ideally serves as the *lower bound* for the Placed(p) growth parameter, because it captures the communication bandwidth requirements independent of the underlying topology constraints, such the low radix mesh. The close similarity in values between Partition(p) and Placed(p) is a strong indication of the *quality* of the task-to-core mapping, *i.e.* the VPR preserves task communication locality.

Although this work does not study the network performance impact of task-to-core mapping in detail, we can evaluate its contribution in the following manner. Let us consider the Min BB mapping to be near-optimal, *i.e.* the best case. It is safe to estimate the lower bound (worst case) on the mapping quality using Random mapping algorithm, which simply assigns each task to a distinct randomly chosen core location. One does not expect the Random algorithm to preserve any communication locality, which the measurements in Table 7.2 demonstrate. The data compares average communication distance D_{avg} on a 2D mesh for Min BB *vs* Random mapping algorithms for 64 and 256 task applications. The best case algorithm exposes inherent communication locality, while the Random algorithm produces the same average communication distance independent of the application.

The numbers produced by the Random algorithm are not accidental. The average communication distance for the *uniformly distributed* traffic on a (n, k)-mesh is $\frac{2}{3} \times n \times k$. Therefore, on a 8×8 2D mesh, the average distance is $\frac{2}{3} \times 2 \times 8 = 5.33$, and for the 16×16 mesh it is 10.66. Table 7.2 also compares the Rent's parameter p and the average communication distance as another illustration of locality gap between the near-optimal Min BB and worst-case Random algorithms. D_{avg} and p are significantly different for Min BB and Random for applications with local communication patterns, such as `gtc3-64` or `cactus-64`. As applications increase their communication range, such as `pmemd-64` or `lbmhd`, the values for D_{avg} and p becomes indistinguishable from the uniform random traffic.

Name	D_{avg}		Dimension Order			MATC		
			Min BB		Random	Min BB		Random
	Min BB	Random	nominal	$N(1, 0.21)$	nominal	nominal	$N(1, 0.21)$	nominal
gtc3-64	1.09	5.44	0.96	0.90	0.33	0.95	0.89	0.36
cactus-64	1.73	5.28	0.91	0.89	0.35	0.90	0.89	0.38
fvcam_2d	2.61	5.32	0.53	0.51	0.32	0.60	0.60	0.35
lbmhd-64	3.36	5.52	0.53	0.51	0.37	0.61	0.60	0.39
pmemd-64	5.24	5.42	0.31	0.29	0.33	0.36	0.35	0.37
gtc2	1.12	11.02	0.98	0.94	0.16	0.98	0.94	0.18
mdh2d	2.02	10.84	0.83	0.81	0.17	0.89	0.87	0.18
cactus-256	2.12	10.40	0.79	0.78	0.18	0.85	0.83	0.17
madbench1	3.31	10.63	0.34	0.31	0.17	0.41	0.38	0.18
slu-256	5.72	10.86	0.22	0.22	0.18	0.25	0.24	0.19
lbmhd	5.81	10.60	0.28	0.25	0.18	0.34	0.33	0.19
madbench2	5.83	10.42	0.25	0.25	0.18	0.25	0.25	0.19
paratec-256	10.66	10.67	0.19	0.17	0.19	0.19	0.18	0.19

Table 7.3: Simulation result comparison for Minimal BB *vs* Random task mapping.

How does the mapping quality affect the actual network performance? Table 7.3 shows the saturation injection rate R for each application mapped by Min BB and Random algorithms. For Min BB, the table contains the saturation injection rate for the nominal (no PVT variations) network and the one with stochastic variations $C_{0.21}$. This helps to compare the performance degradation due to stochastic PVT variations with the potential degradation due to poor task-to-core mapping. The Random mapping turns the applications with mostly local communication pattern, such as `gtc3-64` or `cactus-64`, into the uniform random pattern and dramatically reduces their saturation injection rate from around 0.9 down to 0.3 on 64 core applications. Similarly, on 256 core applications with local communication pattern, the saturation rate is reduced from 0.9 down to 0.17, which is a significant performance degradation. This is the expected result. These numbers are consistent with uniform random traffic on a mesh (Figure 7.18). 8×8 mesh has the topology bound of 0.5, and 16×16 has a bound of 0.25 (Section 6.1). The actual measurements of 0.3 and 0.17 are reasonably close to the router bounds for their respective mesh sizes. The actual saturation rate numbers are lower than the bounds because they reflect communication load imbalance and limits of routing algorithms.

When the application communication pattern is global and approaches that of a uniform random pattern, such as such as `pmemd-64` or `lbmhd`, so does its

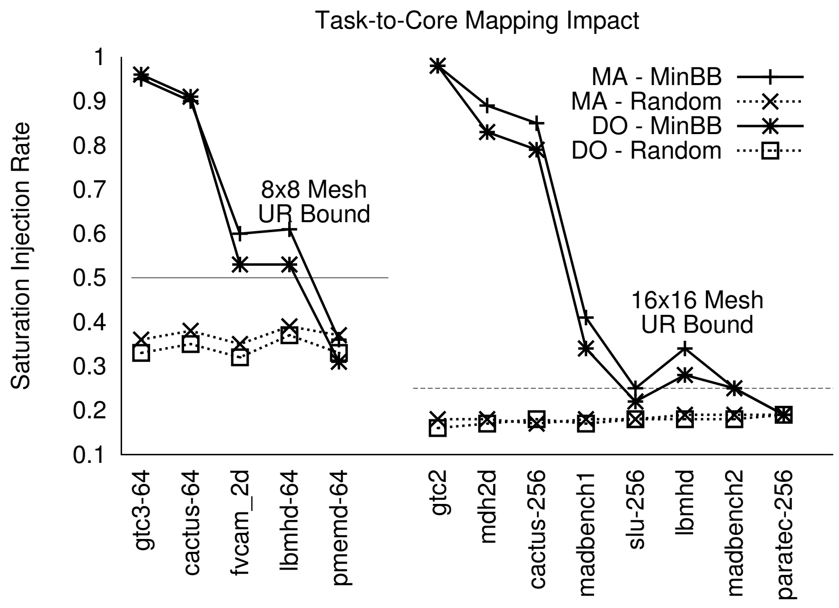


Figure 7.18: Network saturation injection rate comparison with Min BB and Random task mappings.

saturation injection rate. For these applications, the mapping quality is not as critical, since the saturation rate does not degrade significantly on Min BB *vs* Random. However, the impact of the mapping quality should not be underestimated. With the Random mapping, the injection saturation rates obtained from DO and MATC are essentially identical and match the performance of the uniform random communication pattern. In contrast, with Min BB placement, the saturation rate shows an improvement with MATC *vs* DO, particularly for larger 256 core applications. This clearly highlights the importance of a good, task-to-core mapping that preserves application's communication locality.

Figure 7.14 highlights the trend that the applications with higher average communication distance D_{avg} benefit the most from adaptive routing, while mostly local applications do not show a significant benefit due to their limited path diversity. In contrast, Table 7.3 demonstrates that the local applications depend mostly on mapping quality to deliver the performance on par with the expectations.

7.6 Summary

We develop a theoretical foundation for network performance degradation due to PVT variations by introducing a new PVT variation bound that complements the classical topology, routing and switching bounds, which are used today to evaluate application performance. The chapter continues by addressing the core subject of this work: the way to use the adaptive routing to compensate for PVT-induced network performance variations. The NoC architecture abstracts complex parametric variations as core/router tiles with heterogeneous performance and exposes the variations to a routing algorithm as network congestion. Left uncompensated, these variations create communication *load imbalance* that results in a significant degradation of network saturation bandwidth.

We demonstrate with NoC simulation that our simple MATC router simultaneously compensates for load imbalance due to PVT variations and the application communication topology, and reduces the bandwidth degradation by a factor of 2–3x and the standard deviation of saturation bandwidth distribution often by an order of magnitude. The Minimal Adaptive Total Congestion (MATC) router improves the saturation injection rate by 5 to 30% over Dimension Order router, which minimizes the impact of stochastic and systematic on-chip performance variations. These advantages can be turned into smaller design guard-bands for a significant reductions in the implementation area and power consumption. By combining smaller design guard-bands with dynamic DVS/DFS techniques that widen system operating range, the adaptive routing can create the most economical system solution applicable to a wide range of applications.

Section 7.4 illuminates the performance advantages of heterogeneous multicore architectures that are at the foundation of our work. We show 40% difference in the network saturation bandwidth between homogeneous architectures, where all core/router tiles operate at the chip-wide worst-case performance, and the heterogeneous architectures with Voltage Frequency Islands.

Section 7.5 compares the network performance impact of task-to-core mapping quality with that of PVT-induced variations. The analysis demonstrates that task-to-core mapping must preserve application-specific communication locality in order to deliver the network saturation bandwidth close to application routing bound. The mapping quality is the most critical for the applications with local communication pattern, but plays a relatively insignificant role in the applications with global communications whose average packet latency resembles that of the uniform random traffic.

Chapter 8

Network Performance Analysis

Simulation is an accurate but costly method of evaluating network performance of applications with irregular, asymmetric communication topologies and complex load balance profiles. These applications have communication patterns that do not lend themselves to a simple paper and pen analysis. However, we need a static analysis methodology that can help designers to estimate the potential performance gains from adaptive *vs* oblivious routing algorithms for real multitasking applications, not just symmetric, synthetic communication patterns.

We have developed two types of such analysis that relates the saturation bandwidth, the key metric in this work, to the communication locality for arbitrary task graphs. The first analysis method is based on Rent's rule [85], and it enables a designer to quickly and efficiently estimate network topology saturation bandwidth for a particular application communication graph. The second analysis maps a task communication graph into a Multi Commodity Flow problem and estimates routing algorithm bounds of saturation bandwidth — an improvement on the topology bounds. With the *ideal* adaptive routing algorithm, one can expect 10%–60% gains in saturation bandwidth in our High Performance Computing benchmarks.

8.1 Topology Bounds

Application communication patterns can be described with spatial and temporal characteristics as discussed in Section 6.2. Here, we focus on spatial communication characteristics and define a model that enables a designer to efficiently estimate the application performance for a given network topology. Locality has been investigated previously in [76] with a complex analytical model, but we strive toward a simpler *empirical* model that captures performance of an *arbitrary* communication graph.

Traditionally, the routing algorithm performance has been studied using synthetic communication patterns shown in Table 8.1[34]. These patterns span the space from router friendly to adversarial, where the latter requires greater effort from the router to balance the packets across the network links. They are simple to analyze, since all nodes exhibit exactly the same behavior and symmetric communication load balance across the network. In contrast, real multi-tasking

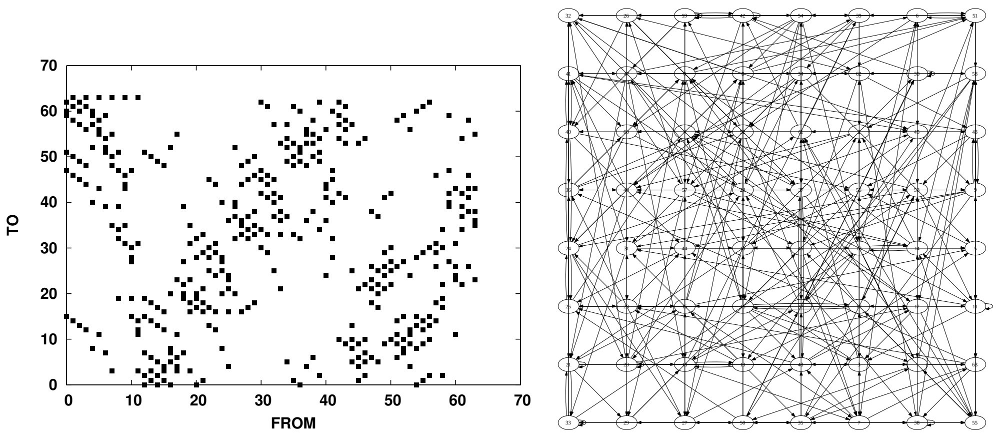
Name	Description
$NN(i)$	**Nearest-neighbor** communication pattern exhibits locality specified by parameter i. $i \in [1, N)$ represents static communication hop distance from the packet source to its destination. For example, $NN(1)$ forms a mesh. This is a router friendly communication pattern with low bandwidth requirements and near-optimal load balance across the network. One expects high network saturation bandwidth, and only a minimal increase in average packet latency at the saturation point.
TOR	**Tornado** communication pattern creates a “twister-like” packet flow on the network. For example, on 2D network, if (x, y) is the address of the packet source, then $(N-1-x, N-1-y)$ is its destination. This is trivially extensible to higher dimensional symmetric network topologies. Although this pattern is symmetric, it has poor load-balance because it creates a highly congested hot-spot in the center of the network at coordinate $(N/2, N/2)$. Tornado is “adversarial” communication pattern that tests router’s ability to load balance packets.
$UR(i)$	**Uniform Random Destination** communication pattern creates a naturally load balanced packet flow because every source selects a destination at random for each packet to be transmitted. This pattern uses *uniformly* distributed selection, but other “skewed” distribution are also possible, such as those targeting destinations at a particular topological distance i from the source. Parameter i can be also used to set the upper bound on the communication distance to control communication locality. Notice that *randomness* implies the spatial and temporal load balance across the network, which is not present in the corresponding $NN(i)$ instance.
R	**Reduce** communication pattern comprises multiple sources and a single destination that becomes a natural bandwidth bottleneck. Even with fair router switching, the packet sources closer to the destination node receive a greater fraction of the switch bandwidth than the distant sources. A routing algorithm has no way of fixing the network load imbalance, since it is forced by the communication topology. However, it may be able to remedy unfair bandwidth allocation between the packet sources with a QoS scheme [86].

Table 8.1: Synthetic, characteristic communication patterns for routing algorithm research.

applications present a more challenging, uneven and asymmetric landscape for routing algorithms. Thus, it is important to study the real applications in addition to the synthetic communication patterns to uncover corner cases.

Our benchmarks represent a collection of communication patterns from local to global, and each one typically comprises several synthetic patterns (Table 8.2). This locality range and complexity makes the benchmark a good choice to study analytically and through network simulation. Notice that the node degree (d_{avg}) is insufficient to estimate the average communication distance D_{avg}, the measure of locality, nor the load balance. For example, Figure 8.1 shows the graph connectivity matrix and communication topology on a 2D mesh network for `lbmhd-64` kernel. Although the connectivity graph is sparse ($d_{avg} = 6$), the locality optimizing task-to-core assignment algorithm creates many connections that span at least half the mesh (Figure 8.1(b)). Communication load imbalance is clear from a simple visual inspection.

In order to understand and interpret routing algorithm performance for these applications, we develop a simple metric that captures the spatial nature of their communication patterns. Borrowing from VLSI, we propose to use Rent's rule analysis to efficiently compute the degree of communication locality of a task graph and also the *growth rate* of its communication bandwidth requirements.



(a) Connectivity Matrix (b) 2D Mesh Task Assigment

Figure 8.1: Spatial communication properties of `lbmhd-64` application.

Name	D_{avg}	N	d_{avg}	Placed (K, p)	Partition (K, p)	Saturat. Rate R
gtc3-64	1.09	64	2.2	(2.91, 0.26)		1
cactus-64	1.73	64	4.5	(3.41, 0.30)	(3.31, 0.29)	1
fvcam_2d	2.62	64	14.4	(3.48, 0.47)	(4.14, 0.39)	1
lbmhd-64	3.37	64	6	(3.52, 0.51)	(3.33, 0.55)	1
pmemd-64	5.24	64	63	(3.45, 0.66)	(3.73, 0.62)	0.64
gtc2	1.12	256	3.1	(4.38, 0.26)	(1.74, 0.14)	1
mdh2d	2.02	256	4	(5.63, 0.31)	(2.74, 0.43)	1
cactus-256	2.12	256	5	(5.75, 0.32)		1
madbench1	3.31	256	13.3	(5.09, 0.45)	(3.90, 0.41)	0.80
slu-256	5.72	256	30.9	(4.66, 0.57)	(3.54, 0.64)	0.49
lbmhd	5.81	256	6	(5.57, 0.53)	(4.17, 0.60)	0.49
madbench2	5.83	256	39.1	(6.25, 0.49)	(5.40, 0.58)	0.53
paratec-256	10.6	256	255	(5.08, 0.68)	(4.38, 0.73)	0.27

Table 8.2: A summary of HPC kernels and their Rent's parameters.

The Rent's rule came from an empirical observation at IBM of the relationship between the number of pins (bandwidth W) at the boundaries of integrated circuit designs and the number of internal components (area A), such as logic gates or standard cells [85]. This relationship can be described by two parameters K and p:

$$W = KA^p \tag{8.1}$$

The pin count growth rate $p \in [0, 1]$ defines the number of pins connecting to the device of area A. $p < 0.5$ generally implies that the growth in the number of pins is slower than the perimeter surrounding the chip area A, and thus the communication is mostly local, nearest neighbor. In contrast $p \geq 0.5$ implies more global, longer distance communication. [42] has formulated a relationship between the Rent's parameters of a VLSI circuit and expected wire lengths.

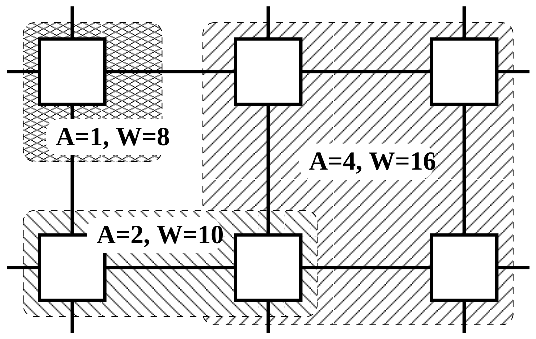
In studying spatial communication patterns of real multiprocessor applications, we apply Rent's rule to extract two parameters (K, p) for each task communication graph. These parameters efficiently summarize application global bandwidth requirements. They enable us to efficiently compute the saturation bandwidth on a particular network topology, which itself is a graph that can be analyzed with Rent's rule.

Define Rent's parameters (K_n, p_n) for the network topology graph. Then the bandwidth W_n that the network can support in and out of a cluster of A processor/router nodes is described by

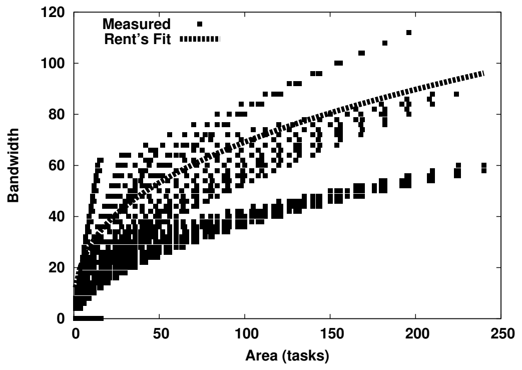
$$W_n = K_n A^{p_n} \tag{8.2}$$

K_n represents the duplex bandwidth of a single $(A = 1)$ network router, and p_n represents the connectivity in a particular network topology. For example, consider a 2D mesh shown on Figure 8.2(a). To compute its Rent's parameters, one must partition the topology into increasingly smaller sub-graphs and record the total capacity of the channels that cross the partitions. The Rent's parameters can be obtained through a simple curve fit as shown on Figure 8.2(b). The approximate 2D mesh parameters are $(K_n, p_n) = (6, 0.37)$. They reflect the nearest neighbor communication topology $(p_n < 0.5)$ and the fact that the single node bandwidth is less than the expected $K_n = 8$ due to the finite size of the mesh.

The application communication topology can be described in the similar manner with parameters (K_a, p_a). K_a represents the maximum bandwidth that an average task consumes and produces and must be $\leq K_n$. The network topology bound and task execution (*e.g.* high compute to communication ratio) reduce the actual consumed bandwidth from its maximum K_a down to RK_a, where $R \in [0, 1]$



(a) Iterative partitioning



(b) Curve fit to estimate parameters.

Figure 8.2: Method to estimate rent's parameters for 2D mesh.

is the Saturation Injection Rate. The injection rate is the fraction of the total network bandwidth utilized by an application. No task can exceed RK_a injection bandwidth because this bandwidth saturates the network and results in a sharp increase in average packet latency. To determine the net saturation bandwidth bound for an application, solve the following optimization problem:

$$\begin{aligned} &\text{Maximize } R \text{ s.t.} \\ &\forall_{A\in[1,N]} RK_a A^{p_a} \quad \leq \quad K_n A^{p_n} \end{aligned} \tag{8.3}$$

Essentially Equation 8.3 states that application will saturate the network with the maximum injection rate R, such that the application bandwidth requirements do not exceed the capacity of the network topology. Since Rent's rule defines bandwidth as monotonically increasing with graph size, which is not always true in practice, to estimate the least upper bound of the saturation injection rate of a particular task communication graph of size A, it suffices to compute the maximum injection rate at the *graph bisection*:

$$R = \frac{K_n \left(\frac{A}{2}\right)^{p_n}}{K_a \left(\frac{A}{2}\right)^{p_a}} = \frac{K_n}{K_a}\left(\frac{A}{2}\right)^{(p_n - p_a)} \tag{8.4}$$

Table 8.2 shows the results of Rent's analysis and the saturation injection rate estimations for our set of HPC applications.

We use the following "Partition" and "Placed" methods to estimate Rent's parameters. (1) The simplest way to compute Rent's parameters (K, p) involves recursive minimal bi-partitioning of the task graph. This methods does not require application tasks to be mapped onto a network. It can provide the best case — the theoretical lower bound — on the bandwidth growth parameter p, since after the tasks are mapped, the 2D network topology imposes its own additional locality constraints. (2) To obtain a more realistic estimate of Rent's parameters and of the saturation injection rate bound R, the task graph must first be mapped onto a network of cores. The tasks were placed onto the 2D mesh network by the simulated annealing algorithm in VPR tool, which minimizes the average communication distance [16]. To compute the "Placed" (K, p) parameters, we recursively bisected the network mesh itself and then measured bandwidth requirement of the application sub-graphs residing in the corresponding network partitions.

Table 8.2 shows a strong correlation between the Rent's parameters obtained with "Placed" and "Partition" methods. The reason why Partition parameters are not strictly the lower bound of the Placed parameters is that they were obtained through a curve fit of Equation 8.1, which has inverse correlation between K and p. Therefore, the only way to accurately compare the two estimates would be to simultaneously fit Placed and Partitioned curves, ensuring that Partition(K) = Placed(K). This is impractical and unnecessary, as the Rent's growth parameter p for Placed and Partition estimates are very close. This indicates that the simulated annealing placement algorithm exposed locality in each communication pattern (further details in Section 7.5).

Table 8.2 also shows the average packet latency (D_{avg}) in router hops. As expected, there is a direct correlation between Rent's growth parameter p and the average communication distance D_{avg}. Using Equation 8.4, we estimated Saturation Injection Rate $R \in [0, 1]$ for each task graph on 2D mesh. Recall that R refers to the largest fraction of the total network bandwidth that a particular communication topology can utilize. As expected, communication locality D_{avg} and saturation injection rate are inversely correlated because as packets travel longer distances in the network, they consume bandwidth from more NoC communication channels and routers, saturating them and increasing congestion.

Rent's parameters provide a convenient *approximation* to the communication capacity of the network and the application requirements. They allow a designer to quickly estimate saturation bandwidth topology bound for an application specific communication pattern. Additionally, they help to estimate overall network channel utilization U:

$$U = \left(\prod_{A \in \left[1, \frac{N}{2}\right]} R \frac{K_a}{K_n} A^{(p_a - p_n)} \right)^{\frac{2}{N}} \tag{8.5}$$

The channel utilization is a geometric mean of bandwidth utilization for application sub-graphs of sizes 1 through $\frac{N}{2}$ tasks. Figure 8.3 illustrates the source of under-utilization in the network with the bandwidth growth parameter p_n that does not match that of the application p_a. Since the topology bound is agnostic to a routing algorithm in that it includes *all* possible paths, this Rentian communication load imbalance cannot be fixed with an adaptive routing.

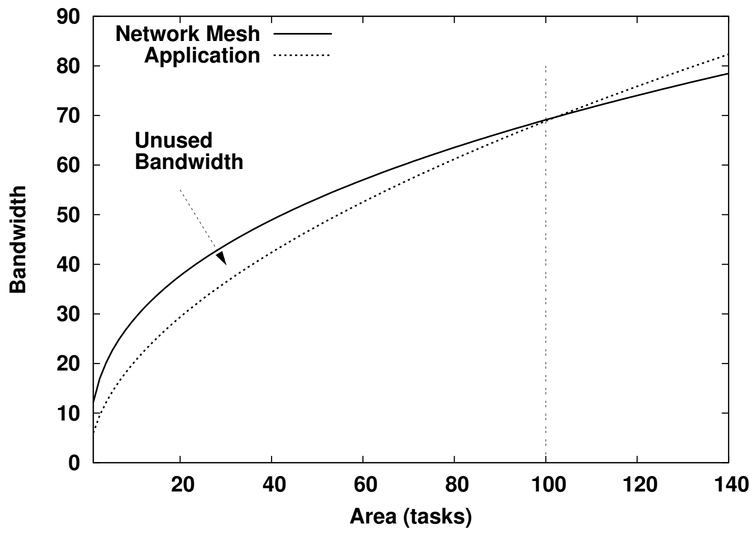


Figure 8.3: Mismatch between network and application Rent's parameters results in underutilized channel bandwidth.

8.2 Router Bounds

We can refine the topology bound with a tighter routing bound, which is specific to a particular routing algorithm and application task-to-core mapping. While the topology bound includes all network paths, the routing bound restricts the paths to only those within a routing algorithm's domain. With routing bounds, we can estimate the network performance advantages of adaptive algorithms *vs* the oblivious ones and determine the limits of the adaptive routing to tolerate performance variations. These bounds also allow us to evaluate the quality of results from NoC simulations in Chapter 7.

The network routing bounds correspond to the volume of the packet flow that saturates at least one critical channel or a router, and permits no further traffic injection. To compute these bounds, we can express the routing problem as Multi-Commodity Maximum Flow (MCMF) problem. Although network packets are discrete entities, a continuous MCMF is a good approximation for the network steady state behavior [34]. We model each task-to-task communication as an independent commodity flow from its source to its destination task, *i.e.* a flow from the core on which the source task is mapped to the core running the destination task. Multiple independent flows share common network resources — channels and routers — that impose capacity constraints. For each task-to-task packet flow, the MCMF contains only a single source and a single destination. Thus a feasible solution to MCMF would respects all packet flows, and maximizing the objective function, the total injected flow, would reveal the network saturation bandwidth bound. To obtain this bound for a specific algorithm and an application, the key to the MCMF problem formulation is to constrain every source-destination flow to only those nodes and edges that a particular router could legitimately use to deliver the packet.

We define the following terms and variables to formalize our MCMF problem. A network $G_N = (V_N, E_N)$ is a graph comprising router/core node set V_N connected with communication channels in set E_N. Each node $u \in V_N$ has ingress and outgress capacities C_u^s and C_u^t, the maximum bandwidth with which a core can push or pull the traffic into/from the network router. Each network channel $(u, v) \in E_N$ has the capacity $C_{(u,v)}$. To simplify the problem formulation, let us assume that network router internal throughput exceeds that of inter-router channels, and thus a router is never the bottleneck in the network. It is simple to extend the problem to include router performance, but the problem will grow in size significantly without revealing any new insight into the routing bound.

The application task graph $G_A = (V_A, E_A)$ includes a set of tasks V_A and unidirectional inter-task flow edges E_A. Each edge $(i, j) \in E_A$ represents a flow that utilizes a collection of network routers and channels along its routed path. An edge has an application defined property, the flow frequency $k_{(i,j)}$ that represents a fraction of the total output bandwidth of task i that is transmitted to task j (Section 5.3). This implies:

$$\forall i \sum_{(i,j) \in E_A} k_{(i,j)} = 1 \tag{8.6}$$

A task-to-core mapping $M : V_A \mapsto V_N$ is required to execute a task graph on a multi-processor system. Given this mapping, a routing algorithm defines a valid path or a set of valid paths that a flow can take for each source and destination pair. Ignoring the order of nodes, a path can be specified as a set of network router nodes $P \subseteq V_N$. A routing algorithm $\mathcal{R}$ can be defined as a relation that maps a pair of source and destination nodes onto a set of valid paths:

$$PSET_{\mathcal{R}} : (V_N \times V_N) \mapsto (P_1, P_2, \ldots) \tag{8.7}$$

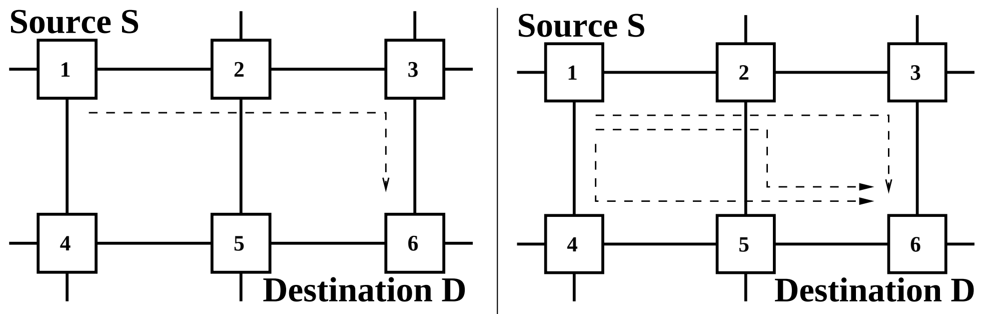
An oblivious algorithm such as Dimension Order defines only a single path for each source and sink node combination, while an adaptive algorithm may produce a larger set of paths. To formulate the Multi Commodity Max Flow problem, define $V^{\mathcal{R}}_{(u,v)}$ and $E^{\mathcal{R}}_{(u,v)}$ as sets of all possible nodes and edges respectively that a traffic from router u could traverse on the path to router v:

$$V^{\mathcal{R}}_{(u,v)} = \bigcup PSET_{\mathcal{R}}(u,v) \tag{8.8}$$

$$E^{\mathcal{R}}_{(u,v)} = \{(p,w) : p,w \in V^{\mathcal{R}}_{(u,v)} \wedge (p,w) \in E_N\} \tag{8.9}$$

Figure 8.4 provides an example of the way these sets are defined for Dimension Order and Minimal Adaptive routers on a 2D mesh. The adaptive algorithm has a significantly greater path diversity.

Next, we define a set of non-negative variables to represent the utilized network capacity. Define s_u to be the amount of traffic injected into the network by the tasks mapped on core u, which is the sum of all packet flows emanating from u to



- $V^{DO}_{(S,D)} = \{1, 2, 3, 6\}$
- $E^{DO}_{(S,D)} = \{(1, 2), (2, 3), (3, 6)\}$

Dimension Order

- $V^{MA}_{(S,D)} = \{1, 2, 3, 4, 5, 6\}$
- $E^{MA}_{(S,D)} = \{(1, 2), (2, 3), (1, 4), (2, 5), (3, 6), (4, 5), (5, 6)\}$

Minimal Adaptive

Figure 8.4: $V^{\mathcal{R}}_{(u,v)}$ and $E^{\mathcal{R}}_{(u,v)}$ for Dimension Order and Minimal Adaptive routers.

their appropriate destinations. The bandwidth consumed by flow $(i, j) \in E_A$, is then $k_{(i,j)} \times s_u$, where u is the router that task i is mapped to ($M[i] = u$). Let t_v be the amount of traffic pulled out of the network by the flow destination router v, which is equal to the sum of all inflows, $\sum_{(i,j)\in E_A \wedge M[j]=v} s_{M[i]} \times k_{(i,j)}$. Let $f_{(i,j)}(u, v)$ be the bandwidth consumed by flow $(i, j) \in E_A$ through edge $(u, v) \in E_N$.

With the terms and variables defined, we are ready to state the capacity and flow preservation constraints for our MCMF problem. The following capacity constraints apply to the sources, sinks and network edges:

$$\forall u \in V_N \quad s_u \leq C^s_u \tag{8.10}$$

$$\forall u \in V_N \quad t_u \leq C_u^t \tag{8.11}$$

$$\forall (i,j) \in E_A \quad f_{(i,j)}(u,v) \leq \begin{cases} \infty & \text{if } (u,v) \in E^{\mathcal{R}}_{(M[i],M[j])} \\ 0 & \text{otherwise} \end{cases} \tag{8.12}$$

$$\forall (u,v) \in E_N \qquad \sum_{(i,j)\in E_A} f_{(i,j)}(u,v) \leq C_{(u,v)} \tag{8.13}$$

Equations 8.10 and 8.11 represent the capacity constraints at the source and destination nodes. Equation 8.12 states that there can be no traffic from task i to task j through edge (u,v) unless it is a part of its routing path. Lastly, Equation 8.13 defines edge capacity constraint.

To simplify the expression of the flow preservation constraints, we define a special variable:

$$S_u^{(i,j)} = \begin{cases} s_u k_{(i,j)} & \text{if } M[i] = u \\ \\ 0 & \text{otherwise} \end{cases} \tag{8.14}$$

$S_u^{(i,j)}$ specifies the amount of traffic injected into the network by network node u for flow (i,j), which can only be nonzero if u is the source of the flow. The flow preservation constraints are defined for each network node in V_N:

$$\begin{array}{l} \forall u \in V_N \\ \forall (i,j) \in E_A \quad \displaystyle\sum_{(v,u)\in E_N} f_{(i,j)}(v,u) - \sum_{(u,w)\in E_N} f_{(i,j)}(u,w) + S_u^{(i,j)} = 0 \\ M[j] \neq u \end{array} \tag{8.15}$$

$$\forall u \in V_N \quad \sum_{(i,j)\in E_A \wedge M[j]=u \wedge (v,u)\in E_N} f_{(i,j)}(v,u) + S_u^{(i,j)} - t_u = 0 \tag{8.16}$$

The constraint in Equation 8.15 applies to each network node and packet flow, as long as the flow does not terminate in that same node. The Equation 8.16 applies to each network node and the flows that terminate there.

With previously stated capacity and flow preservation constraints, maximizing the sum of injected traffic from all sources results in a traffic load that saturates the network and delivers all the flows from their sources to the respective destinations. The upper bound on the saturation injection rate for routing algorithm $\mathcal{R}$ ranges from $\overline{R_1^{\mathcal{R}}}$ to $\overline{R_2^{\mathcal{R}}}$, where each is the value of the MCMF objective function:

$$\overline{R_1^{\mathcal{R}}} = \text{Maximize} \frac{|V_N|s}{\sum_{u\in V_N} C_u^s} \tag{8.17}$$

$$\overline{R_2^{\mathcal{R}}} = \text{Maximize} \frac{\sum_{u\in V_N} s_u}{\sum_{u\in V_N} C_u^s} \tag{8.18}$$

The objective function in Equation 8.18 allows each task to emit traffic at its *own* maximum rate s_i, and thus represents an *overly optimistic* bound on the saturation injection rate. Internal synchronization and dependencies may force the tasks to produce traffic at the *same* steady state rate. This implies that $s_1 = s_2 = \ldots = s_{|V_N|} = s$ and creates a *pessimistic* upper bound on the saturation injection rate shown in Equation 8.17. The actual application-specific bound is

			Dimension Order		Minimal Adaptive		Rent's
Name	D_{avg}	N	Sim	$\overline{R_1^{DO}} - \overline{R_2^{DO}}$	Sim	$\overline{R_1^{MA}} - \overline{R_2^{MA}}$	Bound
gtc3-64	1.09	64	0.96	0.98 – 1.00	0.95	0.98 – 1.00	1
cactus-64	1.73	64	0.91	0.91 – 0.95	0.90	0.91 – 0.95	1
fvcam_2d	2.62	64	0.53	0.67 – 0.84	0.60	0.74 – 0.87	1
lbmhd-64	3.37	64	0.53	0.50 – 0.78	0.61	0.84 – 0.90	1
pmemd-64	5.24	64	0.31	0.41 – 0.48	0.36	0.46 – 0.52	0.64
gtc2	1.12	256	0.98	0.92 – 1.00	0.98	0.99 – 1.00	1
mdh2d	2.02	256	0.83	0.67 – 0.98	0.89	1.00 – 1.00	1
cactus-256	2.12	256	0.79	0.71 – 0.94	0.85	0.84 – 0.96	1
madbench1	3.31	256	0.34	0.40 – 0.74	0.41	0.54 – 0.81	0.8
slu-256	5.72	256	0.22	0.25 – 0.43	0.25	0.25 – 0.44	0.49
lbmhd	5.81	256	0.28	0.29 – 0.51	0.34	0.48 – 0.64	0.49
madbench2	5.83	256	0.25	0.36 – 0.49	0.25	0.49 – 0.54	0.53

Table 8.3: Routing and topology bound *vs* simulation results.

likely to fall somewhere in between. This range points to the expected performance (Table 8.3), and quantifies the expected improvement in saturation bandwidth from adaptive *vs* oblivious routing (Table 8.4), which will be analyzed later.

The stated MCMF optimization problem can be solved with Linear Programming simplex algorithm, which performs well for most application graphs as long as the size of the problem remains manageable. In the worst-case, the number of variables in the problem include $|V_N|$ sources, $|V_N|$ sinks, and $|E_N| \times |V_N|^2$ edge flow variables. There are at most $|V_N| \times |V_N|^2$ flow preservation constraints one for each flow in each network node, and $2|V_N| + |E_N|$ capacity constraints for each source, sink and edge. In practice, as Equation 8.12 shows, many of the edge flow variables can be eliminated from the problem as long as the particular flow

			Routing Bounds		**MA** *vs* **DO** Improv	
Name	D_{avg}	N	DO	MA	$\overline{R_1^{MA}}/\overline{R_1^{DO}}$	$\overline{R_2^{MA}}/\overline{R_2^{DO}}$
gtc3-64	1.09	64	0.98 – 1.00	0.98 – 1.00	1	1
cactus-64	1.73	64	0.91 – 0.95	0.91 – 0.95	1	1
fvcam_2d_r1	2.62	64	0.67 – 0.84	0.74 – 0.87	1.1	1.03
lbmhd-64	3.37	64	0.50 – 0.78	0.84 – 0.90	1.68	1.15
pmemd-64	5.24	64	0.41 – 0.48	0.46 – 0.52	1.12	1.09
gtc2	1.12	256	0.92 – 1.00	0.99 – 1.00	1.08	1
mdh2d	2.02	256	0.67 – 0.98	1.00 – 1.00	1.49	1.02
cactus-256	2.12	256	0.71 – 0.94	0.84 – 0.96	1.18	1.02
madbench1	3.31	256	0.40 – 0.74	0.54 – 0.81	1.35	1.09
slu-256	5.72	256	0.25 – 0.43	0.25 – 0.44	1	1.03
lbmhd	5.81	256	0.29 – 0.51	0.48 – 0.64	1.66	1.26
madbench2	5.83	256	0.36 – 0.49	0.49 – 0.54	1.36	1.11

Table 8.4: The expected saturation bandwidth improvement for Minimal Adaptive *vs* Dimension Order routing algorithm.

does not utilize an edge. This typically brings the problem size down and enables an efficient computation of routing bounds for many interesting application communication topologies.

We evaluated the routing bounds for two important algorithms: Dimension Order and Minimal Adaptive Routing. The construction of their $V^{\mathcal{R}}_{(u,v)}$ and $E^{\mathcal{R}}_{(u,v)}$ sets is illustrated on Figure 8.4. Table 8.3 shows the routing bounds for the application graphs sorted by their size and average communication distance for Minimal Adaptive (MA) and Dimension Order (DO) routing algorithms. Instead of a single bound, a pair $\overline{R^{\mathcal{R}}_1}-\overline{R^{\mathcal{R}}_2}$ illustrates the range of saturation injection rates. For comparison, the table also presents the simulation results from Chapter 7, and the Rent's rule topology bound estimates from Section 8.1. The data shows the correlation between the simulation results, and analytical routing and topology bounds on a 2D mesh.

Two important trends can be observed. As the average communication distance increases, the saturation injection rate R — the maximum fraction of the total network bandwidth that an application can utilize — decreases for both 64 and 256 nodes applications. Table 8.4 shows that Minimal Adaptive Routing has higher saturation bandwidth *vs* the oblivious Dimension Order routing algorithm due to its ability to better utilize network path diversity. As expected, applications with mostly local communication (*e.g.* `gtc3-64` or `gtc2`) do not show significant improvements, but as the communication distance grows so do the gains from adaptive routing, which exploits greater network path diversity. The improvements are not monotonic with communication distance, however, because they depend on a number of other factors such as inherent load imbalance in the application due to its communication properties and its task-to-core mapping. For example, `slu-256` is a large application task graph with communication pattern that is not significantly different from uniform random destination. Therefore, the network traffic is approximately well balanced, and an adaptive algorithm cannot be expected to show much improvement in saturation bandwidth.

8.3 Summary

Network-on-Chip is a distributed system consisting of interconnected core/router tiles, whose performance depends on its topology and cross-section bandwidth, as well as the application that executes on the multi-core NoC system. Spatial and temporal application communication properties such as locality and load balance determine the network saturation bandwidth and the average packet latency.

This chapter defines two analysis methods to quickly evaluate spatial communication patterns of an arbitrary application task graph, and predict its saturation bandwidth bounds dictated by the underlying network topology and the routing algorithm. The results in Tables 8.3 and 8.4 demonstrate high prediction accuracy of our analysis methods *vs* the simulation results in Chapter 6. The Multi-Commodity Max Flow routing model predicts an average improvement of 10-60% in the network saturation bandwidth for an ideal minimally adaptive algorithm compared over the oblivious Dimension Order router.

Chapter 9

Conclusion

As semiconductor manufacturing approaches the limits on transistor dimension scaling and power dissipation, the growth in parametric variations and faults makes it increasingly difficult to deliver the expected die yield using only design time techniques. Currently, Design for Manufacturing (DFM) is the main instrument to manage the yield by increasing layout guard-bands. Circuit designers also use techniques such as adaptive body biasing, supply voltage, and error correction in application certain domains, together with multiple worst-case Process Voltage Temperature corner analysis to attain the desired system parametric yield. These methods result in over-engineered systems that waste area, power and performance.

Due to the lack of a common language between semiconductor manufacturers, layout engineers, circuit designers, and architects that would encompass all four system metrics: area, delay, power and yield — the effective cross layer optimization has so far not been demonstrated. The clean abstractions between these design layers, which are responsible for the historic success of the VLSI industry and performance scaling, are presently its biggest impediment.

The semiconductor device sizes preclude reliable and accurate manufacturing at their own process node, pushing the devices back to larger geometries, a process generation or two. Every extra effort toward fault and variation reliability is costly and perhaps impossible without a diminishing rate of return. Consider that the latest complex, restrictive and costly DFM rules have not been followed by a commensurate increase in performance and power efficiency that we expect from Moore's Law scaling.

Process-Voltage-Temperature (PVT) variations turn a nominally homogeneous many-core die into cores with heterogeneous performance. A distributed Voltage Frequency Island (VFI) architecture can abstract the variations and faults as core/router tile performance (*e.g.* router throughput and latency), that has the potential for a graceful rather than abrupt system performance degradation. A Network-on-Chip is a naturally distributed asynchronous system that can integrate hundreds of these cores in a scalable manner and does not force each core to give up its own PVT-induced operating point for the chip-wide common worst case. Similar to asynchronous logic, a NoC of regular and redundant core VFIs can potentially deliver the average rather than the worst case system performance with power efficiency and fault tolerance [30, 73]. Before this work, VFI architectures have been discussed and hailed for their power/performance, variation and fault resilience [92]. However, those advantages and opportunities have not been demonstrated or quantified as we have done here.

The key to realize the potential performance gains of VFI heterogeneous architectures is run-time adaptivity: task-to-core mapping and adaptive network routing that strives to optimally map application resource requirements onto heterogeneous cores and communication fabric. If observed core/router performance abstracts the PVT variations, mapping and routing can operate *introspectively* without outside performance characterization. By adapting to application-specific behavior on a heterogeneous platform, the run-time system in the same exact manner can handle a variety of effects: static Process faults and variations; dynamic Voltage, Temperature, device aging variations; and application-specific computation and communication load imbalance.

With the gains from adaptive run-time techniques, it is no longer necessary to strive for the maximum, the most expensive last 10–20% in the layout and circuit design. Instead, our systematic techniques mitigate a variety of faults and variations more effectively and comprehensively than layout and circuit DFM. We show that the gains from adaptive routing can be translated into parametric die yield improvements and smaller DFM guard-bands.

This work investigates core sparing and network routing. Sparing is among the most powerful static yield management techniques, particularly well suited in a large collection of redundant components. Based on the developed yield and die cost models, core sparing asymptotically reduces the die cost from $O(A^3)$ to $O(A^{\frac{1}{2}})$, and it is more cost effective than layout and circuit redundancy. Given a fixed total die area, our analysis outcome favors the greater number of smaller unreliable cores to a fewer larger reliable ones. This points to the limitations, and to some degree the futility, of growing DFM and circuit design margins in the future process generations. Figure 9.1 illustrates the simple intuition behind this result. As a unit of fault management and resilience shrinks in size, from a larger die to a smaller core, its yield increases dramatically. Based on our analysis, the smallest efficiently manageable unit of resiliency for sparing, mapping the computation, and routing the communication traffic — the processes required to optimize price/performance of a heterogeneous architecture — is a core/router tile. We neither have the abstractions nor the resources to individually manage smaller components on a system level. According to our model, making circuits and modules within a core more resilient increases the core size and reduces its functional density. This wastes area, power and performance but does not provide an opportunity to manage the redundant resources flexibly, which further increases the system cost.

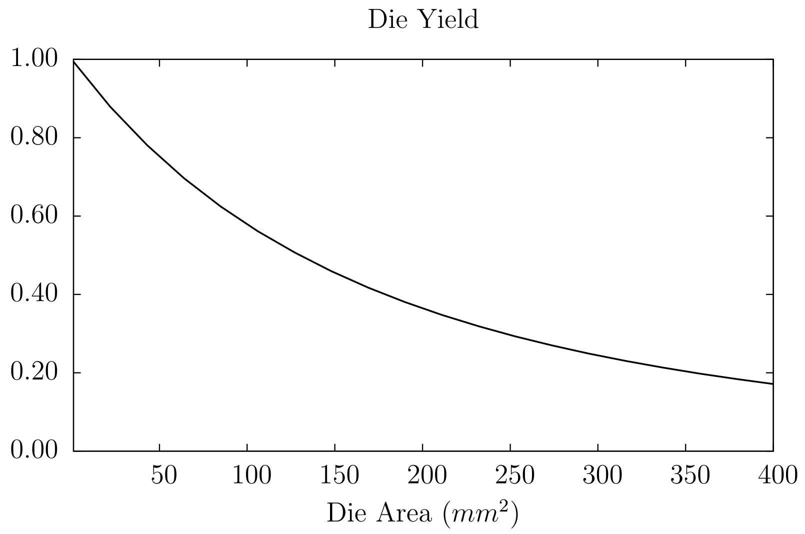


Figure 9.1: The relationship between die area and yield.

This work also focuses on understanding the potential of adaptive network routing. Many researchers identify the benefits of adaptive routing on homogeneous on-chip networks, but we seek to illustrate that routing can additionally minimize the performance impact of PVT variations, which has positive implications for system manufacturing yield and power consumption. Adaptive network routing is required for core sparing because communication traffic must be routed around the faulty and disabled cores. More critically, it combats network load imbalance, a result of PVT on-die heterogeneity and application communication topology. Adaptive routing compensates for all these effects simultaneously and ideally delivers the average rather than the worst case network performance.

Although this work does not focus on task-to-core mapping or location aware task placement in detail, one cannot underestimate the impact of mapping on network performance. Our experiments have demonstrated that mapping quality is critical to preserve communication locality inherent in an application in order to deliver the network performance comparable to its topology bound. Adaptive routing can help the application performance to approach the network topology bound itself by effectively exploiting network path diversity. Figure 9.2 illustrates the relationship between the mapping quality, adaptive routing and network performance for applications with different spatial communication patterns ranging from mostly local to global. Applications with local communication patterns benefit the most from high quality placement. However, they show almost no performance improvement from adaptive *vs* oblivious routing due to their short communication paths and thus limited path diversity. On applications with mostly global communication traffic, where a packet traverses on average half of the network diameter, the role of placement quality is diminished. In contrast, adaptive routing offers significant performance gains by taking advantage of greater path diversity in longer distance communication.

By exposing PVT variations as network router/channel throughput and latency, adaptive routers can compensate for parametric variations using nothing more than the flow control, which enables each router to observe and measure its

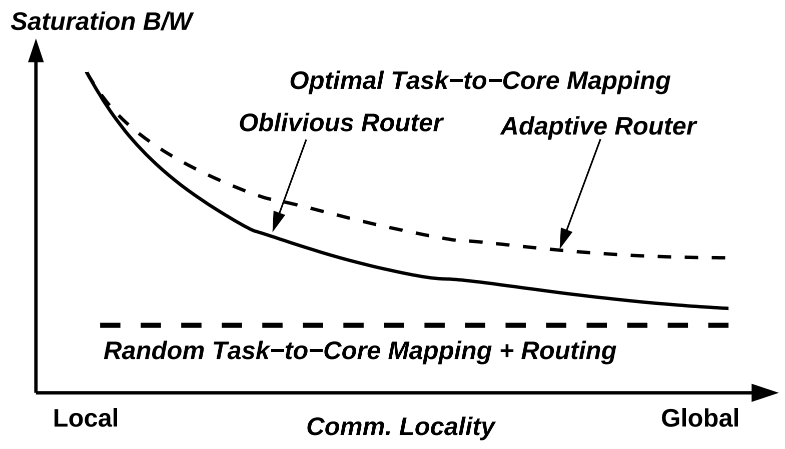


Figure 9.2: Relationship between the mapping quality and routing results.

neighbor's performance. With stochastic and systematic variations, the developed MATC router improves the expected saturation bandwidth 5–30% relative to the oblivious Dimension Order router. These results were achieved with a simple *minimal* adaptive routing algorithm with an adjacent router congestion metric. By exploring several algorithms and adaptive channel selection functions, we found that the actual metric is not critical. Any congestion indicator such as total or partial buffer occupancy, or the output router channel throughput produce approximately equivalent network performance. The only critical algorithm feature is its routing adaptivity in all network topology directions. Adaptive routing minimizes the network performance degradation due to PVT variations and can thus be used to reduce over-engineering guard-bands in multi-core architectures.

By treating cores as units of fault and variation tolerance, core sparing and routing provide a simple and consistent way to compensate for static and dynamic performance variations and faults instead of isolated ad hoc DFM and circuit solutions. By exposing the variations and faults as observable core/router performance, adaptive run-time techniques can address a wide range of problematic effects, including those that are unknown or impossible to identify today.

Chapter 10

Future Directions

10.1 Refining Die Yield Models

Chapter 4 presented the analytical die yield and cost models for core sparing, circuit module redundancy, and layout device over-engineering. These model point to core sparing as the most economical redundancy option due to its flexibility and improved yield as the cores shrink in size. The modeling assumes that the improvements in core yield from Design-for-Manufacturing (DFM) layout techniques scales linearly and continuously with the design margins. However, intuitively, the impact of DFM depends on individual rules and their combinations as applied to a site on a die. To fully understand the trade-off between a system with a large number of small unreliable cores and the one with fewer hardened cores, we need a more detailed and accurate DFM yield models. DFM rules affects a range of semiconductor device features, and thus must be applied in a more focused manner: some rules may have a marginal impact on the defect yield but a significant impact on process parametric variations, while others are critical to ensure functional, defect-free operation only.

Unfortunately, a more detailed bottom up analysis of increasingly complex DFM layout rules is outside of realm of academic research. At the same time, a better understanding of their impact on transistor and interconnect functional and parametric yield can result in significant reductions in wasted area, performance and power. Chapter 7 demonstrated that high-level adaptive routing can tolerate considerable performance variation in Network-on-Chip, and we expect similar optimistic results from variation aware task-to-core mapping, as suggested by some sources [103]. We call for a revised approach to DFM that can correlate the individual rules and their parameters with the the effects on system performance variations and fault probabilities. The architects, designers, layout engineers can then determine the magnitude of variations that the should be tolerate in a cost-efficient manner with their respective techniques. Ultimately, this will result in a methodology to search for the optimal balance between power, performance and yield across multiple system implementation levels.

10.2 Task-to-Core Mapping

This work argues for a high-level system approach to tolerating and compensating PVT induced performance variations and focuses on one part of this strategy: the adaptive routing techniques. Dynamic task-to-core mapping for multi-core architectures with heterogeneous performance is the other very critical component of this strategy that can improve system variation tolerance. Mapping is the process of assigning tasks to execute on the cores that maximally match the task's computational resource requirements. It has the potential for power efficiency, performance improvement, and most critically reductions in manufacturing margins and costs by delivering the average case, closer to nominal, system computational performance.

The mapping also has a dramatic impact on network performance as discussed in Section 7.5. In contrast with systems with a four to eight cores, large multi-core architectures with hundreds of cores are performance heterogeneous and are impacted by communication locality. For these systems, task-to-core mapping must also preserve that locality in addition to maintaining core load balance. The network performance results in this work were obtained with a mapping that simply strove to minimize average communication distance between the cores.

Although the task-to-core assignment for the *minimum* average communication distance does not guarantee the best network performance, it is likely to be close. The mapping algorithm that only considers the distance between communicating task attempts to reduce the number of links utilized by a typical communication flow. It does not, however, consider communication direction or which specific physical links are utilized, and whether the assignment can be modified to reduce the discrepancy between saturated and unsaturated links. We call for bandwidth oriented task-to-core assignment that would consider inter-router channel utilization. This algorithm would be able to take advantage of asymmetry in the application graph when mapping onto a symmetric communicating fabric such as a mesh.

Independent of the details of a specific task assignment algorithm, we believe that it must be a distributed algorithm to avoid a single point of failure in an environment where a system comprises a large number of unreliable components. We are inspired by [138], which presented a distributed simulated annealing algorithm for FPGA placement, as an example of an algorithm that solves a complex global mapping problem without a centralized control.

10.3 Using Architecture Knobs

Chapters 2 and 3 discussed the way PVT variations can turn a functionally homogeneous multi-core architecture into a performance heterogeneous computation fabric with core-level voltage and frequency domains. The power and performance advantages of this approach have already been demonstrated for several small scale architectures, and the advantages will only magnify with hundreds of cores per die and a greater PVT diversity. Chapter 7 has demonstrated the way adaptive routing can compensate for PVT induced performance variations across the NoC. The adaptive routing algorithm, as presented, only *reacts* to the dynamically changing on-chip performance profile. However, the run-time adaptive processes can go one step further and utilize the architecture knobs, such as core dynamic voltage and frequency scaling, to navigate the operating range of each core and router. By utilizing these knobs, the routing and mapping form a closed loop system with the hardware. They can react to PVT induced dynamic performance profile as well as *modify* it to optimally match the hardware performance with the application resource requirements. This complex approach holds a promise of further power/performance efficiency improvement, and additional reductions in manufacturing guard-bands and margins.

Bibliography

[1] International Technology Roadmap for Semiconductors, http://www.itrs.net, 2005.

[2] IPM homepage, http://www.nersc.gov/projects/ipm, 2005.

[3] The network simulator: NS-2, http://www.isi.edu/nsnam/ns/, 2008.

[4] F. Adamu-Lema, S. Roy, A. R. Brown, A. Asenov, and G. Roy. Intrinsic parameter fluctuations in conventional mosfets at the scaling limit: a statistical study. In *International Workshop on Computational Electronics (IWCE)*, October 2004.

[5] N. Agarwal, L.-S. Peh, and N. Jha. Garnet: A detailed interconnection network model inside a full-system simulation framework. Technical Report CE-P08-001, Dept. of Electrical Engineering, Princeton University, 2008.

[6] M. Alcubierre, G. Allen, B. Bruegmann, E. Seidel, and W.-M. Suen. Towards an understanding of the stability properties of the 3+1 evolution equations in general relativity. *Physical Review D*, 62:124011, 2000.

[7] M. Amde, T. Felicijan, A. Efthymiou, D. Edwards, and L. Lavagno. Asynchronous on-chip networks. In *IEE*, 2005.

[8] T. Austin, D. Blaauw, T. Mudge, and K. Flautner. Making typical silicon matter with RAZOR. In *IEEE Computer*, volume 37(3), March 2004.

[9] J. Balfour and W. J. Dally. Design tradeoffs for tiled cmp on-chip networks. In *ICS '06: Proceedings of the 20th annual international conference on Supercomputing*, pages 187–198, New York, NY, USA, 2006. ACM.

[10] E. Beigne, F. Clermidy, P. Vivet, A. Clouard, and M. Renaudin. An asynchronous noc architecture providing low latency service and its multi-level design framework. pages 54–63, March 2005.

[11] A. Bender. Milp based task mapping for heterogeneous multiprocessor systems. In *European Design Automation Conference*, pages 283–288, 1996.

[12] M. A. Bender and M. O. Rabin. Scheduling Cilk multithreaded parallel programs on processors of different speeds. In *Proceedings of the twelfth annual ACM symposium on Parallel algorithms and architectures*, pages 13–21, July 2000.

[13] D. Bertozzi, L. Benini, and G. de Micheli. Low power error resilient encoding for on-chip data buses. In *DATE '02: Proceedings of the conference on Design, automation and test in Europe*, page 102, Washington, DC, USA, 2002. IEEE Computer Society.

[14] D. Bertsekas and R. Gallagher. *Data Networks: Second Edition.* Prentice-Hall, Inc., 1992.

[15] D. P. Bertsekas and J. N. Tsitsiklis. *Introduction to Probability.* Athena Scientific, 2002.

[16] V. Betz and J. Rose. VPR: A new packing, placement and routing tool for FPGA research. In W. Luk, P. Y. Cheung, and M. Glesner, editors, *Field-Programmable Logic and Applications*, pages 213–222. Springer-Verlag, Berlin, 1997.

[17] B. Biklsma. Asynchronous network-on-chip architecture performance analysis. Master's thesis, Delft University of Technology, 2005.

[18] D. Boning and S. Nassif. Models of process variations in device and interconnect, 2000.

[19] S. Borkar. Thousand core chips: a technology perspective. In *DAC '07: Proceedings of the 44th annual conference on Design automation*, pages 746–749, New York, NY, USA, 2007. ACM.

[20] S. Y. Borkar. Private communication, 2006.

[21] J. Borrill, J. Carter, L. Oliker, and D. Skinner. Integrated performance monitoring of a cosmology application on leading hec platforms. In *ICPP '05: Proceedings of the 2005 International Conference on Parallel Processing*, pages 119–128, Washington, DC, USA, 2005. IEEE Computer Society.

[22] K. A. Bowman and J. D. Meindl. Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration. In *IEEE JSSC*, Feb 2002.

[23] N. Campregher, P. Y. K. Cheung, G. A. Constantinides, and M. Vasilko. Analysis of yield loss due to random photolithographic defects in the interconnect structure of fpgas. In *FPGA '05: Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, pages 138–148, New York, NY, USA, 2005. ACM Press.

[24] A. Canning, L. W. Wang, A. Williamson, and A. Zunger. Parallel empirical pseudopotential electronic structure calculations for million atom systems. *J. Comput. Phys.*, 160(1):29–41, 2000.

[25] K. Cao. Predictive technology model (http://www.eas.asu.edu/ ptm/).

[26] Y. Cao and L. T. Clark. Mapping statistical process variations toward circuit performance variability: an analytical modeling approach. In *DAC '05: Proceedings of the 42nd annual conference on Design automation*, pages 658–663, New York, NY, USA, 2005. ACM.

[27] Y. Cao, C. Hu, A. Kahng, and D. Sylvester. Improved estimates of process variation impact on deep submicron circuit performance. (unpublished).

[28] Y. Cao, H. Qin, R. Wang, P. Friedberg, A. Vladimirescu, and J. Rabaey. Yield optimization with energy-delay constraints in low power digital circuits. In *International Conference Electronics Development and Solid State Circuits, Kowloon, Hong Kong*, pages 285–288, 2003.

[29] T. Chen and S. Naffziger. Comparison of adaptive body bias (abb) and adaptive supply voltage (asv) for improving delay and leakage under the presence of process variation. *IEEE Trans. Very Large Scale Integr. Syst.*, 11(5):888–899, 2003.

[30] C.-L. Chou, U. Ogras, and R. Marculescu. Energy- and performance-aware incremental mapping for networks on chip with multiple voltage levels. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27:1866–1879, October 2008.

[31] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition.* The MIT Press, September 2001.

[32] M. F. Crowley, I. David W. Deerfield, T. A. Darden, and I. Thomas E. Cheatham. Molecular dynamics simulations using particle-mesh ewald methods. pages 355–387, 2000.

[33] J. Dai and B. Prabhakar. The throughput of data switches with and without speedup. In *IEEE Infocom*, March 2000.

[34] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[35] W. J. Dally. Performance analysis of k-ary n-cube interconnection networks. *IEEE Transactions on Computers*, 39(6):775–785, 1990.

[36] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. Comput.*, 36(5):547–553, 1987.

[37] W. J. Dally and B. Towles. Route packets, not wires: on-chip inteconnection networks. In *DAC '01: Proceedings of the 38th conference on Design automation*, pages 684–689, New York, NY, USA, 2001. ACM.

[38] K. Datta, M. Murphy, V. Volkov, S. Williams, J. Carter, L. Oliker, D. Patterson, J. Shalf, and K. Yelick. Stencil computation optimization and autotuning on state-of-the-art multicore architectures. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–12, Piscataway, NJ, USA, 2008. IEEE Press.

[39] H. de Man. Bridging the gaps between gigascale integration and nano-scale technology. Presentation at ESSCIRC/ESSDERC, 2006.

[40] A. DeHon, P. Lincoln, and J. E. Savage. Stochastic assembly of sublithographic nanoscale interfaces. 2(3):165–174, September 2003.

[41] H. DeMan. Ambient intelligence: Giga-scale dreams and nano-scale realities. In *Proc of ISSCC, Keynote Speech*, February 2005.

[42] W. Donath. Placement and average interconnection lengths of computer logic. 26:272 – 277, April 1979.

[43] J. Dorsey, S. Searles, M. Ciraula, E. Fang, S. Johnson, N. Bujanos, and R. Kumar. An integrated quad-core opteron processor. In *Solid-State Circuits Conference*, 2007.

[44] J. Dorsey, S. Searles, M. Ciraula, S. Johnson, N. Bujanos, D. Wu, M. Braganza, S. Meyers, E. Fang, and R. Kumar. An integrated quad-core opteron processor. In *ISSCC*, 2007.

[45] A. Dua, N. Bambos, W. Olesinski, H. Eberle, and N. Gura. Backlog aware low complexity schedulers for input queued packet switches. In *HOTI '07: Proceedings of the 15th Annual IEEE Symposium on High-Performance Interconnects*, pages 39–46, Washington, DC, USA, 2007. IEEE Computer Society.

[46] J. Duato. On the design of deadlock-free adaptive routing algorithms for multicomputers: design methodologies. In *PARLE '91: Proceedings on Parallel architectures and languages Europe : volume I: parallel architectures and algorithms*, pages 390–405, New York, NY, USA, 1991. Springer-Verlag New York, Inc.

[47] J. Duato, A. Robles, F. Silla, and R. Beivide. A comparison of router architectures for virtual cut-through andwormhole switching in a now environment. In *Symposium on Parallel and Distributed Processing*, pages 240 – 247, April 1999.

[48] J. Duato, S. Yalamanchili, and L. Ni. *Interconnection Networks: an Engineering Approach*. IEEE Computer Society, 1997.

[49] J. Duato, S. Yalamanchili, and L. Ni. *Interconnection Networks: An engineering approach*. Published by Morgan Kaufmann, 2002.

[50] T. Dumitras and R. Marculescu. On-chip stochastic communication, 2003.

[51] W. Eatherton. The push of network processing to the top of the pyramid. In *keynote addres at Symposium on Architectures for Networking and Communication Systems*, October 2005.

[52] M. Elgebaly and M. Sachdev. Efficient adaptive voltage scaling system through on-chip critical path emulation. In *ISLPED '04: Proceedings of the 2004 international symposium on Low power electronics and design*, pages 375–380, New York, NY, USA, 2004. ACM.

[53] D. Ernst, S. Das, S. Lee, T. A. David Blaauw, T. Mudge, N. S. Kim, and K. Flautner. Razor: Circuit-level correction of timing errors for low-power operation. *IEEE Micro*, 24(6):10–20, November 2004.

[54] A. V. Ferris-Prabhu. *Introduction to Semiconductor Device Yield Modeling.* Artech House Publishers, August 1992.

[55] D. J. Frank, R. H. Dennard, E. Nowak, P. M. Solomon, Y. Taur, and H.-S. P. Wong. Device Scaling Limits of Si MOSFETs and Their Application Dependencies. In *Proceedings of the IEEE*, volume 89, March 2001.

[56] J. Friedrich, B. McCredie, N. James, B. Huott, B. Curran, E. Fluhr, G. Mittal, E. Chan, Y. Chan, D. Plass, S. Chu, H. Le, L. Clark, J. Ripley, S. Taylor, J. Dilullo, and M. Lanzerotti. Design of the power6 microprocessor. In *ISSCC*, 2007.

[57] R. Goering. Do recommended dfm rules really matter? November 2007.

[58] M. Gomez, N. Nordbotten, J. Flich, P. Lopez, A.Robles, J. Duato, T. Skeie, and O. Lysne. A routing methodology for achieving fault tolerance in direct networks. In *IEEE Trans on Computers*, pages 400–415, 2006.

[59] J. Gregg and T. W. Chen. Optimization of individual well adaptive body biasing (iwabb) using a multiple objective evolutionary algorithm. In *ISQED '05: Proceedings of the 6th International Symposium on Quality of Electronic Design*, pages 297–302, Washington, DC, USA, 2005. IEEE Computer Society. Colorado State.

[60] J. Gu, S. S. Sapatnekar, and C. Kim. Width-dependent statistical leakage modeling for random dopant induced threshold voltage shift. In *DAC '07: Proceedings of the 44th annual conference on Design automation*, pages 87–92, New York, NY, USA, 2007. ACM.

[61] A. Gupta, R. Chauhan, V. Menezes, V. Narang, and R. H.M. A robust level-shifter design for adaptive voltage scaling. *VLSI Design, International Conference on*, 0:383–388, 2008.

[62] T. Hanyu, T. Ike, and M. Kameyama. Self-checking multiple-valued circuit based on dual-rail current-mode differential logic. In *29th IEEE International Symposium on Multiple-Valued Logic*, pages 275–279, 1999.

[63] J. C. Harden and N. R. S. II. Architectural yield optimization for wsi. *IEEE Trans. Comput.*, 37(1):88–110, 1988.

[64] M. D. Harmanci, N. P. Escudero, Y. Leblebici, and P. Ienne. Providing qos to connection-less packet-switched noc by implementing diffserv functionalities. In *International Symposium on System-on-Chip*, pages 37 – 40, November 2004.

[65] M. Horowitz and W. J. Dally. How scaling will change processor architecture. In *International Solid-State Circuits Conference*, pages 132–133, February 2004.

[66] J. Hu and R. Marculescu. Dyad: Smart routing for networks-on-chip. In *Design Automation Conference*, June 2004.

[67] W. Huang, E. Humenay, K. Skadron, and M. R. Stan. The need for a full-chip and package thermal model for thermally optimized ic designs. In *ISLPED '05: Proceedings of the 2005 international symposium on Low power electronics and design*, pages 245–250, New York, NY, USA, 2005. ACM.

[68] E. Humenay, D. Tarjan, and K. Skadron. Impact of parameter variations on multi-core chips. In *Workshop on Architectural Support for Gigascale Integration*, June 2006.

[69] S. Inaba. Iedm technical digest, 2001.

[70] E. İpek, S. A. McKee, R. Caruana, B. R. de Supinski, and M. Schulz. Efficiently exploring architectural design spaces via predictive modeling. *SIGPLAN Not.*, 41(11):195–206, 2006.

[71] J. Jackson. Jobshop-like queueing systems. *Management Science*, 10:131 – 142, 1963.

[72] P. K. Jana. Multi-mesh of trees with its parallel algorithms. *J. Syst. Archit.*, 50(4):193–206, 2004.

[73] W. Jang and A. J. Martin. Soft-error robustness in qdi circuits.

[74] K. Jeong, A. Kahng, and K. Samadi. Quantified impacts of guardband reduction on design process outcomes. In *International Symposium on Quality Electronic Design*, March 2008.

[75] J. H. Jiang, Y. H. Min, and C. L. Peng. Fault-tolerant systems with concurrent error-locating capability. *Computer Science Technology*, 18(2):190–200, 2003.

[76] K. L. Johnson. The impact of communication locality on large-scale multiprocessor performance. In *Computer Architecture News*, pages 392–402. ACM, 1992.

[77] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy. Introduction to the cell multiprocessor. *IBM J. Res. Dev.*, 49(4/5):589–604, 2005.

[78] P. Kermani and L. Kleinrock. Virtual cut-through: a new computer communication switching technique. *Computer Networks*, 3:267–286, 1979.

[79] J. Kibarian. Iccad 2007 keynote speech, November 2007.

[80] J. Kim, W. J. Dally, B. Towles, and A. K. Gupta. Microarchitecture of a high-radix router. In *ISCA '05: Proceedings of the 32nd annual international symposium on Computer Architecture*, pages 420–431, Washington, DC, USA, 2005. IEEE Computer Society.

[81] I. Koren and C. M. Krishna. *Fault-Tolerant Systems.* Morgan Kaufmann, 2007.

[82] R. Kumar, K. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen. Processor power reduction via single-isa heterogeneous multi-core architectures. *IEEE Comput. Archit. Lett.*, 2(1):2, 2003.

[83] H. T. Kung and S. Y. Wang. Zero queueing flow control and applications. In *INFOCOM*, pages 192–200, 1998.

[84] Y.-K. Kwok and I. Ahmad. Link contention-constrained scheduling and mapping of tasks and messages to a network of heterogeneous processors. In *International Conference on Parallel Processing*, pages 551–558, 1999.

[85] B. S. Landman and R. L. Russo. On a pin versus block relationship for partitions of logic graphs. C-20:1469 – 1479, December 1971.

[86] J. W. Lee, M. C. Ng, and K. Asanovic. Globally-synchronized frames for guaranteed quality-of-service in on-chip networks. *Computer Architecture, International Symposium on*, 0:89–100, 2008.

[87] X. S. Li and J. W. Demmel. Superlu dist: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Transactions on Mathematical Software*, 29:110–140, 2003.

[88] Z. Lin, S. Ethier, T. S. Hahm, and W. M. Tang. Size scaling of turbulent transport in magnetically confined plasmas. *Phys. Rev. Lett.*, 88(19):195004, Apr 2002.

[89] S.-C. Lo and Y. Li. Numerical simulation of random dopant fluctuation in sub-65 nm metal-oxide-semiconductor field effect transistors. In *MATH'05: Proceedings of the 8th WSEAS International Conference on Applied Mathematics*, pages 272–280, Stevens Point, Wisconsin, USA, 2005. World Scientific and Engineering Academy and Society (WSEAS).

[90] A. Macnab, G. Vahala, P. Pavlo, and L. Vahala. Lattice Boltzmann Model for Dissipative Incompressible MHD. *APS Meeting Abstracts*, pages 1130P–+, Oct. 2001.

[91] H. Mahmoodi-Meimand, S. Mukhopadhyay, and K. Roy. Estimation of delay variations due to random-dopant fluctuations in nano-scaled cmos circuits. In *IEEE Custom Integrated Circuits Conference*, 2004.

[92] R. Marculescu, D. Marculescu, and L. Pileggi. "toward an integrated design methodology for fault-tolerant, multiple clock/voltage integrated systems". In *2004 IEEE International Conference on Computer Design (ICCD'04)*, pages 168–173, 2004.

[93] M. Marsan, A. Bianco, E. Filippi, P. Giaconne, and E. L. adn F Neri. On the behavior of input queueing switch architectures. *European Transactions on Teleommunications (ETT)*, 10:111 – 124, March/April 1999.

[94] A. J. Martin and M. Nystrom. Asynchronous techniques for system-on-chip design. pages 1089–1120, June 2006.

[95] R. McGowen, C. Poirier, C. Bostak, J. Ignowski, M. Millican, W. Parks, and S. Naffziger. Power and temperature control on a 90-nm itanium family processor. *IEEE Journal of Solid-State Circuits*, 41:229–237, January 2006.

[96] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand. Achieving 100% throughput in an input-queued switch. In *IEEE Transactions on Communication*, July 2004.

[97] A. Meixner, M. E. Bauer, and D. J. Sorin. Argus: Low-cost, comprehensive error detection in simple cores. *IEEE Micro*, 28(1):52–59, 2008.

[98] G. Microsystems. G2c547 product brief.

[99] M. Mohiyuddin, M. Murphy, S. Williams, L. Oliker, J. Shalf, and J. Wawrzynek. Hardware/software co-tuning for power efficient scientific computing, 2009.

[100] G. E. Moore. Cramming more components onto integrated circuits. 38(8), 1965.

[101] S. Nassif. Delay variability: sources, impacts and trends. In *Solid-State Circuits Conference, 2000. Digest of Technical Papers. ISSCC.*, pages 368 – 369, 2000.

[102] U. G. Nawathe, M. Hassan, L. Warriner, K. Yen, B. Upputuri, D. Greenhill, A. Kumar, and H. Park. An 8-core 64-thread 64b power-efficient sparc soc. In *ISSCC*, 2007.

[103] U. Y. Ogras, R. Marculescu, P. Choudhary, and D. Marculescu. Voltage-frequency island partitioning for gals-based networks-on-chip. In *Proc. IEEE/ACM Design Automation Conf., San Diego*, June 2007.

[104] P. Oldiges, Q. Lin, K. Petrillo, M. Sanchez, M. Ieong, and M. Hargrove. Modeling line edge roughness effects in sub 100 nanometer gatelength devices. *International Conference on Simulation of Semiconductor Processes and Devices*, pages 131–134, 2000.

[105] L. Oliker, J. Carter, M. Wehner, A. Canning, S. Ethier, A. Mirin, D. Parks, P. Worley, S. Kitawaki, and Y. Tsuda. Leading computational methods on scalar and vector hec platforms. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 62, Washington, DC, USA, 2005. IEEE Computer Society.

[106] M. Orshansky, L. Milor, and C. Hu. Characterization of spatial intra-field gate cd variability, its impact on circuit performance, and spatial mask-level correction. *IEEE Transactions on Semiconductor Manufacturing*, (1):2–11, February 2004.

[107] L. T. Pang. *Measurement and Analysis of Variability in CMOS circuits*. PhD thesis, EECS Department, University of California, Berkeley, Aug 2008.

[108] D. A. Patterson and J. L. Hennessy. *Computer architecture: a quantitative approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[109] W. W. Peterson and E. J. Weldon. *Error-Correcting Codes*. MIT Press, 2 edition, 1972.

[110] D. Pham, S. Asano, M. Bolliger, M. N. Day, H. P. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Riley, D. Shippy, D. Stasiak, M. Suzuoki, M. Wang, J. Warnock, S. Weitzel, D. Wendel, T. Yamazaki, and K. Yazawa. The design and implementation of a first-generation cell processor. pages 184–592 Vol. 1, 2005.

[111] Predictions Software Ltd. Yield modeling, http://www.icyield.com/yieldmod.html, 2005.

[112] J. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits: A Design Perspective; 2nd ed.* Pearson Education, Upper Saddle River, NJ, 2003. Order from outside CERN via Inter Library Loan.

[113] N. H. Z. Radu Marculescu, Umit Y. Ogras. Computation and communication refinement for multiprocessor soc design: A system-level perspective. *ACM Trans. Design Autom. Elect. Syst. Special Issue on Novel Paradigms in System-Level Design*, 11(3):564–592, July 2006.

[114] V. S. P. Rapaka, E. Talpes, and D. Marculescu. Mixed-clock issue queue design for energy aware, high-performance cores. In *ASP-DAC '04: Proceedings of the 2004 conference on Asia South Pacific design automation*, pages 380–383, Piscataway, NJ, USA, 2004. IEEE Press.

[115] N. Sakran, M. Yuffe, M. Mehalel, J. Doweck, E. Knoll, and A. Kovacs. The implementation of the 65nm dual-core 64b merom processor. In *ISSCC*, 2007.

[116] N. Sarkan, M. Yuffe, J. Doweck, E. Knoll, and A. Kovacs. Implementation of the 65nm dual-core 64b merom processor. In *Solid-State Circuits Conference*, 2007.

[117] D. Schroder. Negative bias temperature instability: What do we understand? 47:841–852, June 2007.

[118] P. Sedcole and P. Y. K. Cheung. Parametric yield in fpgas due to within-die delay variations: a quantitative analysis. In *FPGA '07: Proceedings of the 2007 ACM/SIGDA 15th international symposium on Field programmable gate arrays*, pages 178–187, New York, NY, USA, 2007. ACM Press.

[119] J. Shalf, S. Kamil, L. Oliker, and D. Skinner. Analyzing ultra-scale application communication requirements for a reconfigurable hybrid interconnect. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 17, Washington, DC, USA, 2005. IEEE Computer Society.

[120] J. Shalf, S. Kamil, L. Oliker, and D. Skinner. Analyzing ultra-scale application communication requirements for a reconfigurable hybrid interconnect. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 17, Washington, DC, USA, 2005. IEEE Computer Society.

[121] D. Shelepov and A. Fedorova. Scheduling on heterogeneous multicore processors using architectural signatures.

[122] A. Singh, W. J. Dally, A. K. Gupta, and B. Towles. Adaptive channel queue routing on k-ary n-cubes. In *SPAA '04: Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 11–19, New York, NY, USA, 2004. ACM.

[123] A. Singh, W. J. Dally, A. K. Gupta, and B. Towles. Adaptive channel queue routing on k-ary n-cubes. In *SPAA*, June 2004.

[124] A. Singh and M. Marek-Sadowska. Fpga interconnect planning. In *SLIP '02: Proceedings of the 2002 international workshop on System-level interconnect prediction*, pages 23–30, New York, NY, USA, 2002. ACM.

[125] V. Soteriou, N. Eisley, H. Wang, B. Li, and L.-S. Peh. Polaris: A system-level roadmap for on-chip interconnection networks. In *In Proceedings of the 24th International Conference on Computer Design (ICCD)*, October 2006.

[126] V. Soteriou, H. Wang, and L.-S. Peh. A statistical traffic model for on-chip interconnection networks. In *In Proceedings n Proceedings of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, September 2006.

[127] V. Srinivasan, D. Brooks, M. Gschwind, P. Bose, V. Zyuban, P. N. Strenski, and P. G. Emma. Optimizing pipelines for power and performance. In *MICRO 35: Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, pages 333–344, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[128] B. Stackhouse, B. Cherkauer, M. Gowan, P. Gronowski, and C. Lyles. A 65nm 2-billion-transistor quad-core itanium processor. In *Solid-State Circuits Conference*, 2008.

[129] B. Stackhouse, B. Cherkauer, M. Gowan, P. Gronowski, and C. Lyles. A 65nm 2-billion-transistor quad-core itanium processor. In *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*, pages 92–598, 2008.

[130] C. H. Stapper. Improved yield models for fault-tolerant memory chips. *IEEE Trans. Comput.*, 42(7):872–881, 1993.

[131] C. E. Stroud. Yield modeling for majority voting based defect-tolerant vlsi circuits. In *IEEE Southeastcon*, pages 229–236, 1999.

[132] E. Talpes and D. Marculescu. Toward a multiple clock/voltage island design style for power-aware processors, 2005.

[133] J. W. Tschanz, J. T. Kao, S. G. Narendra, R. Nair, D. A. Antoniadis, A. P. Ch, S. Member, and V. De. Adaptive body bias for reducing impacts of die-to-die and within-die parameter variations on microprocessor frequency and leakage. In *IEEE Journal Of Solid-State Circuits*, pages 1396–1402, 2002.

[134] H. Tuinhout. Impact of parametric fluctuations on performance and yield of deep-submicron technologies. In *32nd European Solid-State Device Research Conference*, pages 95 – 102, 2002.

[135] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, and S. Venkataraman. An 80-tile 1.28tflops network-on-chip in 65nm cmos. In *ISSCC*, 2007.

[136] H. Wang, X. Zhu, L.-S. Peh, and S. Malik. Orion: A power-performance simulator for interconnection networks. In *In Proceedings of MICRO 35*, November 2002.

[137] F. Worm, P. Ienne, P. Thiran, and G. D. Micheli. A robust self-calibrating transmission scheme for on-chip networks. *IEEE Trans. Very Large Scale Integr. Syst.*, 13(1):126–139, 2005.

[138] M. G. Wrighton and A. M. DeHon. Hardware-assisted simulated annealing with application for fast fpga placement. In *FPGA '03: Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays*, pages 33–42, New York, NY, USA, 2003. ACM Press.

[139] J. Wu. Unicasting in faulty hypercubes using safety levels. In *International Conference on Parallel Processing*, 1995.

[140] J. Wuu, D. Weiss, C. Morganti, and M. Dreesen. The asynchronous 24mb on-chip level-3 cache for a dual-core itanium-family processor. In *IEEE International Solid-State Circuits Conference, Digest of Technical Papers*, volume 1, pages 488–612, 2005.

[141] Y. Yoshida, T. Kamei, K. Hayaase, S. Shibahara, O. Nishii, T. Hatton, A. Hasegawa, M. Takada, N. Irie, T. Odaka, K. Takada, K. Kimura, and H. Kasahara. A 4320mips four-process core smp/amp with individually managed clock frequency for low power consumption. In *Solid-State Circuits Conference*, 2007.

[142] Y. Yoshida, T. Kamei, K. Hayase, S. Shibahara, O. Nishii, T. Hattori, A. Hasegawa, M. Takada, N. Irie, K. Uchiyama, T. Odaka, K. Takada, K. Kimura, and H. Kasahara. A 4320mips four-processor core smp/amp with individually managed clock frequency for low power consumption. In *ISSCC*, 2007.

[143] H. Yu, I.-H. Chung, and J. Moreira. Topology mapping for blue gene/l supercomputer. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 116, New York, NY, USA, 2006. ACM.

[144] Z. Yu and B. Baas. Implementing tile-based chip multiprocessors with gals clocking styles. In *IEEE International Conference of Computer Design (ICCD)*, pages 174–179, October 2006.

[145] H. Zhang and J. Rabaey. Low-swing interconnect interface circuits. In *ISLPED '98: Proceedings of the 1998 international symposium on Low power electronics and design*, pages 161–166, New York, NY, USA, 1998. ACM.

[146] M. Zhang and N. R. Shanbhag. A cmos design style for logic circuit hardening. In *International Reliability Physics Symposium*, 2005.

[147] M. Zhong. Evaluation of deflection-routed on-chip networks. Master's thesis, KTH, Stockholm, Sweden, 2005.

Appendices

Appendix A

Adaptive Routing and PVT Variations

Saturation Injection Rate Degradation due to Stochastic Variations

Name	D_{avg}	DO μ				DO σ			DO	
		0	0.07	0.14	0.21	0.07	0.14	0.21	A	B
gtc3-64	1.1	0.96	0.92	0.89	0.87	0.016	0.011	0.019	0.41	0.09
cactus-64	1.7	0.90	0.88	0.88	0.87	0.007	0.008	0.013	0.15	0.06
fvcam_2d	2.6	0.48	0.48	0.47	0.47	0.003	0.005	0.008	0.06	0.04
lbmhd-64	3.4	0.49	0.48	0.48	0.47	0.013	0.016	0.016	0.09	0.08
pmemd-64	5.2	0.31	0.30	0.29	0.28	0.008	0.010	0.014	0.14	0.07
gtc2	1.1	0.99	0.97	0.95	0.94	0.007	0.010	0.011	0.26	0.05
mdh2d	2.0	0.77	0.76	0.76	0.75	0.004	0.008	0.009	0.11	0.04
cactus-256	2.1	0.74	0.74	0.74	0.73	0.003	0.006	0.009	0.06	0.04
madbench1	3.3	0.34	0.34	0.32	0.31	0.007	0.013	0.016	0.16	0.08
slu-256	5.7	0.22	0.21	0.21	0.21	0.006	0.009	0.010	0.06	0.05
lbmhd	5.8	0.29	0.28	0.27	0.25	0.006	0.008	0.011	0.18	0.05
madbench2	5.8	0.24	0.24	0.23	0.23	0.002	0.004	0.005	0.05	0.02
paratec-256	10.7	0.18	0.18	0.17	0.16	0.003	0.004	0.004	0.08	0.02
		MATC μ				MATC σ			MATC	
gtc3-64	1.1	0.96	0.92	0.89	0.87	0.016	0.011	0.019	0.41	0.09
cactus-64	1.7	0.90	0.88	0.88	0.87	0.007	0.008	0.013	0.15	0.06
fvcam_2d	2.6	0.48	0.48	0.47	0.47	0.003	0.005	0.008	0.06	0.04
lbmhd-64	3.4	0.49	0.48	0.48	0.47	0.013	0.016	0.016	0.09	0.08
pmemd-64	5.2	0.31	0.30	0.29	0.28	0.008	0.010	0.014	0.14	0.07
gtc2	1.1	0.99	0.97	0.95	0.94	0.007	0.010	0.011	0.26	0.05
mdh2d	2.0	0.77	0.76	0.76	0.75	0.004	0.008	0.009	0.11	0.04
cactus-256	2.1	0.74	0.74	0.74	0.73	0.003	0.006	0.009	0.06	0.04
madbench1	3.3	0.34	0.34	0.32	0.31	0.007	0.013	0.016	0.16	0.08
slu-256	5.7	0.22	0.21	0.21	0.21	0.006	0.009	0.010	0.06	0.05
lbmhd	5.8	0.29	0.28	0.27	0.25	0.006	0.008	0.011	0.18	0.05
madbench2	5.8	0.24	0.24	0.23	0.23	0.002	0.004	0.005	0.05	0.02
paratec-256	10.7	0.18	0.18	0.17	0.16	0.003	0.004	0.004	0.08	0.02

Degradation and performance due to systematic variations

Name	D_{avg}	DO					MATC				
		0	SP(0.90,1.10)	SP(0.80,1.20)	SP(0.70,1.30)	A	0	SP(0.90,1.10)	SP(0.80,1.20)	SP(0.70,1.30)	A
gtc3-64	1.1	0.96	0.97	0.94	0.93	0.05	0.95	0.97	0.94	0.93	0.03
cactus-64	1.7	0.91	0.90	0.89	0.88	0.05	0.90	0.90	0.89	0.87	0.05
fvcam_2d	2.6	0.53	0.54	0.53	0.53	-0.00	0.60	0.59	0.58	0.58	0.03
lbmhd-64	3.4	0.53	0.54	0.54	0.53	-0.00	0.61	0.60	0.60	0.59	0.03
pmemd-64	5.2	0.31	0.31	0.30	0.29	0.03	0.36	0.38	0.38	0.38	-0.03
gtc2	1.1	0.98	0.97	0.95	0.93	0.08	0.98	0.97	0.95	0.93	0.08
mdh2d	2.0	0.83	0.83	0.83	0.82	0.02	0.89	0.89	0.61	0.87	0.03
cactus-256	2.1	0.79	0.81	0.79	0.79	-0.00	0.85	0.86	0.84	0.83	0.03
madbench1	3.3	0.34	0.35	0.37	0.37	-0.05	0.41	0.51	0.50	0.50	-0.15
slu-256	5.7	0.22	0.23	0.23	0.22	-0.00	0.25	0.25	0.24	0.25	-0.00
lbmhd	5.8	0.28	0.29	0.27	0.26	0.03	0.34	0.36	0.36	0.35	-0.02
madbench2	5.8	0.25	0.29	0.28	0.28	-0.05	0.25	0.38	0.38	0.36	-0.18
paratec-256	10.7	0.19	0.19	0.18	0.16	0.05	0.19	0.20	0.20	0.20	-0.02

Name	D_{avg}	MATC Net Improv. vs Nominal DO							MATC Net Improv. vs Var DO					
		static	$N(1, 0.07)$	$N(1, 0.14)$	$N(1, 0.21)$	$SP(0.9, 0.9, 1.1)$	$SP(0.8, 0.8, 1.2)$	$SP(0.7, 0.7, 1.3)$	$N(1, 0.07)$	$N(1, 0.14)$	$N(1, 0.21)$	$SP(0.9, 0.9, 1.1)$	$SP(0.8, 0.8, 1.2)$	$SP(0.7, 0.7, 1.3)$
gtc3-64	1.09	1.00	0.97	0.94	0.93	1.01	0.99	0.96	1.00	0.99	0.99	1.00	1.00	1.00
cactus-64	1.73	1.00	0.99	0.99	0.98	0.99	0.98	0.96	1.00	1.00	0.99	0.99	0.99	0.99
fvcam_2d_region1	2.61	1.12	1.12	1.12	1.12	1.10	1.09	1.09	1.12	1.14	1.17	1.09	1.11	1.10
lbmhd-64	3.36	1.16	1.16	1.15	1.13	1.15	1.14	1.12	1.15	1.18	1.18	1.11	1.11	1.13
pmemd-64	5.24	1.15	1.15	1.13	1.13	1.21	1.22	1.21	1.16	1.18	1.22	1.21	1.25	1.32
gtc2	1.12	1.00	0.98	0.97	0.96	0.98	0.96	0.94	1.00	1.00	1.00	1.00	1.00	1.00
mdh2d	2.02	1.07	1.06	1.05	1.04	1.06	0.73	1.04	1.07	1.08	1.07	1.06	0.74	1.06
cactus-256	2.12	1.09	1.08	1.07	1.06	1.09	1.07	1.06	1.09	1.07	1.07	1.06	1.06	1.06
madbench1	3.31	1.20	1.23	1.17	1.11	1.48	1.46	1.46	1.27	1.24	1.23	1.43	1.34	1.34
slu-256	5.72	1.13	1.13	1.13	1.13	1.16	1.12	1.14	1.12	1.11	1.12	1.10	1.08	1.11
lbmhd	5.81	1.21	1.19	1.17	1.15	1.26	1.25	1.23	1.22	1.25	1.29	1.22	1.32	1.33
madbench2	5.83	1.00	1.00	1.01	1.00	1.52	1.50	1.45	0.96	1.00	1.02	1.30	1.33	1.28
paratec-256	10.66	0.99	0.99	0.97	0.95	1.03	1.03	1.03	1.01	1.04	1.09	1.05	1.11	1.19