Energy Consumption in Networks on Chip: Efficiency and Scaling

by

George B. P. Bezerra

B.S., Electrical Engineering, University of Campinas, Brazil, 2005
M.S., Computer Engineering, University of Campinas, Brazil, 2006

DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Computer Science

The University of New Mexico

Albuquerque, New Mexico

December, 2012

Dedication

To my parents, Albert II and Gladys, for their support, encouragement and the Corvette they're giving me for graduation.

"A bird in hand is worth two in the bush" – Anonymous

Acknowledgments

I would like to thank my advisor, Professor Martin Sheen, for his support and some great action movies. I would also like to thank my dog, Spot, who only ate my homework two or three times. I have several other people I would like to thank, as well.[1]

[1]To my brother and sister, who are really cool.

Energy Consumption in Networks on Chip: Efficiency and Scaling

by

George B. P. Bezerra

B.S., Electrical Engineering, University of Campinas, Brazil, 2005
M.S., Computer Engineering, University of Campinas, Brazil, 2006

PhD., Computer Science, University of New Mexico, 2012

Abstract

Computer architecture design is in a new era where performance is increased by replicating processing cores on a chip rather than making CPUs larger and faster. This design strategy is motivated by the superior energy efficiency of the multi-core architecture compared to the traditional monolithic CPU. If the trend continues as expected, the number of cores on a chip is predicted to grow exponentially over time as the density of transistors on a die increases.

A major challenge to the efficiency of multi-core chips is the energy used for communication among cores over a Network on Chip (NoC). As the number of cores increases, this energy also increases, imposing serious constraints on design and performance of both applications and architectures. Therefore, understanding the impact of different design choices on NoC power and energy consumption is crucial to the success of the multi- and many-core designs.

This dissertation proposes methods for modeling and optimizing energy consumption in multi- and many-core chips, with special focus on the energy used for commu-

nication on the NoC. We present a number of tools and models to optimize energy consumption and model its scaling behavior as the number of cores increases. We use synthetic traffic patterns and full system simulations to test and validate our methods. Finally, we take a step back and look at the evolution of computer hardware in the last 40 years and, using a scaling theory from biology, present a predictive theory for power-performance scaling in microprocessor systems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the last 40 years, the monolithic CPU design has scaled in performance by six orders of magnitude, following the exponential trends dictated by Moore's law. However, the traditional sources of performance improvement—e.g. instruction level parallelism and clock frequency increase—have saturated. Computer architecture design has entered a new era, in which performance is increased by adding more CPUs (or cores) to a chip instead of making larger and faster ones. As transistor density continues to increase fostered by process technology improvement, an exponential growth in the number of cores on a chip is expected in the following years [84].

This shift of the industry towards the multi-core architecture is motivated primarily by energy and power consumption. With the highly complex circuits and high clock frequency of modern billion-transistor CPUs, we have reached the limit of how much power (and heat) can be dissipated on a chip and cooled by air. The increased demand for embedded systems, such as smartphones, tablets and netbooks, has also raised the concern for energy and power consumption. Because these devices rely on limited source of energy, embedded processors are designed to minimize energy consumption in order to increase battery life. Energy is also a major concern in servers

and data centers. Currently, the energy used for operating and cooling servers accounts for about 1% of all the electricity in the United States [56]. A study shows that the energy bill for data centers worldwide more than doubled between 2000 and 2005 [57]. With the increasing demand for Web content, this situation is expected to worsen in the upcoming years. In modern High-Performance Computing (HPC) systems, power is also the biggest constraint. The exascale computing challenge aims at improving the performance of supercomputers by 1000× in 10 years, but current trends in energy consumption for such machines render this goal unattainable [54]. A large investment is being made to increase the energy efficiency of supercomputers, in the US and worldwide, in order to avoid the energy bottleneck in the future generation of HPC machines.

The multi-core architecture is a viable alternative to the traditional monolithic design because parallelization is a more energy-efficient way of improving performance [39]. Because performance of individual cores is not expected to increase significantly in the future, the energy used per core for computation will likely decrease with miniaturization. However, cores need to communicate at increased rates as the system grows in size, which increases the energy consumption used for communication on the Network on Chip (NoC). This energy will vary considerably depending on the behavior of applications and on the locality of their communication patterns, but it will invariably increase, imposing serious constraints on design and performance of both applications and architectures.

This dissertation focuses on the modeling and optimization of energy and power consumption in multi-core chips. We present a number of tools and models to optimize energy consumption and model its scaling, and use synthetic traffic patterns and full system simulations to test and validate our methods. Finally, we take a step back and look at the evolution of computer hardware in the last 40 years and, employing an interdisciplinary approach, devise a theoretical framework for the analysis

of power-performance scaling in microprocessor systems in general.

1.1 Contributions and organization

The remaining of this dissertation is divided into 7 chapters. In this section, we summarize the contents and contributions of each chapter.

Chapter 2. This chapter contains a review of the background material that is required for reading this dissertation. We give an introduction to networks on chip, its basic architecture and topologies. Also included is a review of memory and communication models in parallel architectures, such as shared and distributed memory systems. Finally, we discuss traditional methods for thread-mapping and scheduling used for optimizing runtime and energy consumption in parallel architectures.

Chapter 3. This chapter analyzes the impact of different NoC topologies to the scaling of power consumption. Using a simple model of inter-core communication, we provide an asymptotic analysis of power and performance in network on chip topologies. The results of the theoretical model are then compared with simulations, showing an excellent match between theory and experiment. This chapter concludes with a surprising result: although power and performance may vary widely between topologies, their energy efficiency is approximately the same, and that the traffic pattern is a more important factor to efficiency than topology.

Chapter 4. The most important component to NoC energy consumption is the communication locality of the network traffic pattern. This chapter analyzes the impact of commonly used artificial workloads and proposes a new synthetic traffic pattern based on Rent's rule. This new method has the advantage of

matching observed data and the ability of, by tweaking a single parameter, emulate a continuum of applications with varying degrees of communication locality.

Chapter 5. Here, we take a deeper look into the multi-core architecture and analyze how the execution of applications can be optimized in order to reduce energy consumption. We propose a method for data placement optimization in shared-memory CMPs, which reduces NoC energy consumption by 50% on average on scientific benchmarks, outperforming a state-of-the-art method. Different from other approaches in the literature which are heuristic-based, our method is exact and can be solved in polynomial time. The experiments in this chapter were performed using full-system simulations and 64 cores.

Chapter 6. In this chapter, we use a theoretical framework based on Rent's rule to analyze the minimum possible energy consumption of an application. We show that applications with low Rent's exponents can be highly optimized for low energy consumption, while applications with high Rent's exponent are bound to be energy-inefficient. Our model shows an excellent match with experimental results obtained using full-system simulations.

Chapter 7. This chapter puts together many of the elements developed in the previous chapters and, employing an interdisciplinary approach, proposes an innovative theoretical framework for the study of power-performance scaling in general-purpose microprocessors. This theory, originally inspired by Metabolic Scaling Theory in biology, analyzes power as being determined by the geometry of computer interconnects. The proposed analysis leads to extremely simple laws that describe the scaling of power and performance in microprocessors over a range of several orders of magnitude.

Chapter 8. This chapter concludes the dissertation.

Chapter 2

Background Information

This chapter covers the background information on networks on chip and computer architectures required in the following chapters. Section 2.1 introduces the NoC communication structure and reviews its properties and most common topologies. Section 2.2 reviews the different memory and communication models of parallel architectures. Special focus is given to shared memory systems, which is the dominant architectural organization for multi-core chips. Finally, Section 2.3 introduces the basic ideas of communication locality and, consequently, energy optimization through mapping and scheduling of threads.

2.1 Network on Chip

2.1.1 Overview

Cores need to communicate in order to share load an decrease the running time of applications. The traditional way in which different modules communicate on a chip is by using dedicated buses. This communication architecture can be very

efficient for a small number of devices, but it lacks scalability for large systems. The reason is that, as the system grows, the increased competition for network usage leads to more collisions, and starvation is more likely to occur. Moreover, as more units are attached to the network and the bus grows in size, the capacitance of the wires increases and the energy necessary for switching a bit becomes too high, leading to excessive power consumption. Finally, as capacitance increases, the time necessary for the residual currents to die away in order to perform a new switching also increases, leading to a prohibitively low bandwidth.

To cope with these limitations, the computer architecture research community proposed a new concept called Networks on Chip (NoC). The NoC is a packet-switched interconnection network for Systems on Chip (SoC), in which modules (also called *Intellectual Properties*—IPs) communicate by sending and receiving messages. Figure 2.1 depicts the basic structure of an NoC. In this design, switches deal with temporary storing and forwarding of packets and the implementation of routing and arbitration protocols. The network interface (NI) provides the interface between the IP and the network and is responsible for many functions, such as packetization and de-packetization of messages, clock adaptation, flow control, and implementing the network protocols [26]. It serves as a standard interface between any device and the network, thus allowing for reuse of IPs. When a module needs to communicate, the message to be sent is divided into packets that are sent to a switch, which forwards the packets to another switch, until the message arrives at the appropriate destination.

A SoC can be homogeneous or heterogeneous. In a homogeneous SoC, all IPs are of the same type, such as cores in a Chip Multi-Processor (CMP), used for general-purpose computing. Heterogeneous SoCs are commonly used for application-specific tasks, such as in embedded systems, and have modules of different types. The SoC approach, using an NoC for communication, has many advantages over the traditional interconnect design used in monolithic CPUs. The point-to-point,

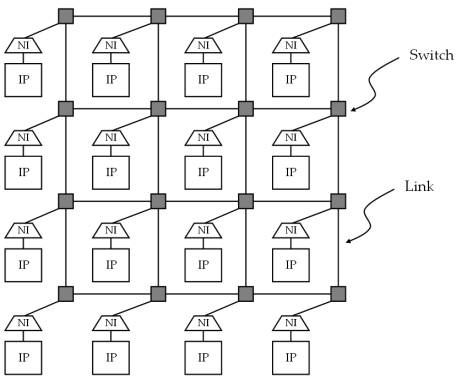


Figure 2.1: Basic structure of an NoC.

dedicated wires of monolithic architectures are among the main drawbacks of the traditional system. These unstructured wires lead to low bandwidth, low duty factor and high power consumption, and also increase the complexity of the design. Based on these considerations, the advantages of a NoC for SoCs were summarized by [24] as follows:

1. *Predictable electrical parameters enable high-performance circuits.* Unstructured wires have parasitic capacitances and crosstalk noise that are difficult to predict. As a result, in order to ensure reliability, very conservative circuits must be used to drive and receive these wires, leading to excessive power consumption. The well structured and predictable wires of a NoC allow for aggressive circuit techniques, which can reduce power dissipation by a factor of ten and increase wire propagation by three times, while also improving bandwidth.

2. *Universal interface facilitates reuse of components.* By introducing a universal interface for IPs and the network, components can be reused in many systems, thus reducing complexity and simplifying circuit implementation.

3. *Design and testing are facilitated.* Since the system is modular and compo-

nents are reused, design and testing of entire systems is mostly concerned with optimization of a regular, generic communication medium with predictable parameters. CAD issues involved in the design of dedicated, customized circuits in specific components, such as wiring routing, are avoided.

4. *Duty factor of the wires is improved.* In traditional chip designs, individual signals must travel as fast as possible to their specific destination, leading to an excessive number of dedicated global wires which are active only 10% of the time, in average. The aggregated flux of information in general-purpose NoCs can provide wire duty factors close to 100%.

5. *Enable the use of fault-tolerant strategies.* With technology scaling and decrease in the voltage usage wires become more susceptible to noise and faults. Eventually, it will be impossible to completely avoid such errors (called upsets) in communication, and the system must be able to deal with them. A NoC architecture can implement error-identification/error-correction protocols that make the system tolerant to faults.

6. *Wire pipelining.* Globally asynchronous protocols allow for wire pipelining, thus increasing bandwidth and making communication independent of latency.

7. *Scalability.* The NoC architecture is scalable; the aggregated bandwidth increases with network size.

2.1.2 NoC Topologies

The NoC depicted in Figure 2.1 uses a 2D mesh topology. However, many other NoC topologies have been proposed, most of which were adapted from the parallel computing world. Some of the most popular NoC topologies are shown in Figure 2.2.

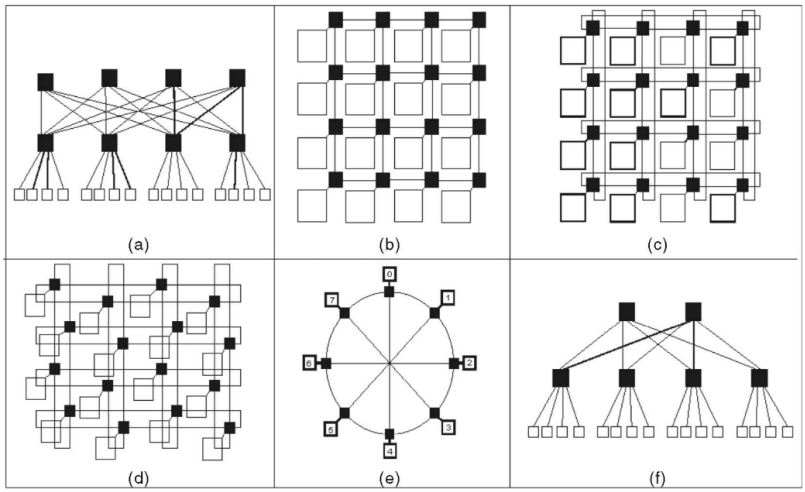


Figure 2.2: NoC topologies. (a) Fat tree, (b) mesh, (c) torus, (d) folded-torus, (e) octagon, (f) butterfly fat tree. (Figure reproduced from [73].)

One of the main properties of a topology is bisection bandwidth scaling. The *bisection width* is the number of wires that must be cut when the network is divided into two equal sets of nodes, and the *bisection bandwidth* is the collective bandwidth over these wires. As more nodes are attached to the network, the larger the volume of communication and the more bandwidth is required. If the network bandwidth does not scale appropriately with the number of nodes, excessive traffic will lead to high message latency and decreased performance. However, networks with high bisection bandwidth will require more routers and more wires per node, which consume considerable area and increase the cost of the system.

Many studies have been dedicated to compare topologies in the NoC literature. Pande *et al.* [74, 73] compared five topologies, studying throughput, latency, energy consumption and area requirements, using nearest-neighbor local traffic with different injection rates. They reported results for system sizes up to 256 cores, showing a trade-off of topologies that provide better throughput and latency but have high power and wiring overhead, versus topologies with lower performance and reduced power consumption. Kreutz *et al.* [60] analyzed bit energy consumption and la-

tency for mesh, torus, and fat tree topologies on 16-node networks, using Romberg integration, Fast Fourier Transform, and an image processing application to generate workloads. They concluded that the fat tree minimizes latency and the mesh topology consumes the least energy.

Boroni and Concer [11] compared ring, spidergon, and mesh topologies under uniform random, homogeneous sources and destinations, and hotspot traffic patterns. They simulated networks ranging from 8 to 32 cores, measuring throughput and latency as a function of injection rate. The spidergon performed the best. Boroni et al. [10] extended this work, studying systems up to 64 cores and adding the crossbar topology, which slightly outperformed the spidergon. Rahmati *et al.* [78] analyzed latency and power consumption for mesh and WK-recursive topologies with 16 cores and uniform traffic. WK-recursive was found to be superior to mesh in terms of latency and power consumption for low traffic, but the two were similar under heavy traffic. A similar comparison by Suboh *et al.* [87] included the spidergon, also concluding that the WK-recursive network had the best performance. Koohi *et al.* [55] and Mirza-Aghatabar *et al.* [69] analyzed 6×6 mesh and torus networks with uniform random, hot-spot, nearest-neighbor, and first matrix transpose traffic. They measured latency, power consumption, and throughput, finding that the torus has higher power dissipation than the mesh but performs better in terms of power/throughput.

Topologies are usually compared in terms of power, throughput, and latency, as a function of packet injection rate and under different traffic patterns, but an important aspect of a topology is how easily it can be implemented on chip. Since chips are two-dimensional, topologies such as mesh and torus are more naturally suited to physical implementation on a die. Topologies have also key impact on the routing algorithms that can be used. Because the routers, and not the wires, are the main bottleneck in terms of latency and bandwidth in an NoC, routing must be simple enough so that the routing decision is made as quickly as possible. A simple

yet deadlock-free algorithm is *xy*- or dimension-order routing, which is designed for regular topologies, such as mesh and torus. Because of these advantages, these two topologies are usually considered the best candidates for NoC [47].

2.2 Memory and Communication Models in CMP

How do cores communicate? What kind of information is exchanged among them and how is it structured? The answer to these questions depends on the particular memory and communication models adopted in a given CMP architecture. In this section, we review the main communication models for multi-processing. We will give special emphasis to the models suited to implementation on chip.

2.2.1 Models of Parallel Architectures

There are two main models of parallel architectures: shared memory and distributed memory. In shared memory architectures, multiple processors share a same memory address space. As will be explained in Section 2.2.2, in these systems communication between processors occurs implicitly by cache coherence. Figure 2.3 depicts the basic structure of a *centralized shared memory* architecture, where multiple processors access a single physical memory. This architecture is also called *uniform memory access* (UMA), from the fact that all processors have a uniform latency to memory.

In *distributed memory* systems, the memory is physically distributed among processors and each processor has private access to its local memory. In such architectures, communication occurs explicitly by message passing. Figure 2.4 shows the basic structure of a distributed memory system. Notice that the address space is private to each memory.

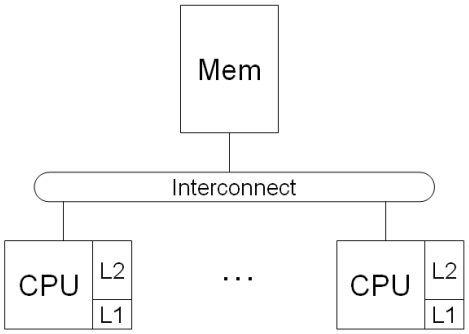


Figure 2.3: Centralized shared memory architecture, also called UMA. All CPU nodes access a single physical memory. L1 and L2 correspond to private level 1 and level 2 caches.

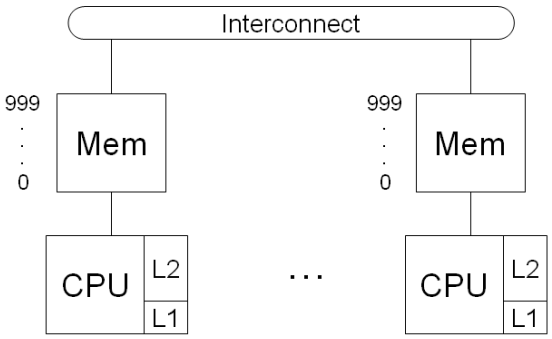


Figure 2.4: Distributed memory architecture. Memory address spaces are independent and private to each CPU.

Other paradigms exist between these two extremes. The *distributed shared memory* architecture is a shared memory system in which memory is physically distributed. Each processor is associated with a local memory, but all memories in the system use the same address space, as depicted in Figure 2.5. Such architectures are also called *non-uniform memory access* (NUMA), because local memory access has lower latency than remote memory access.

The next model is the *non-uniform cache access* architecture (NUCA). The NUCA architecture is a variation of the UMA system, where the higher level cache is also shared (Figure 2.6). More specifically, the L2 caches are distributed but they

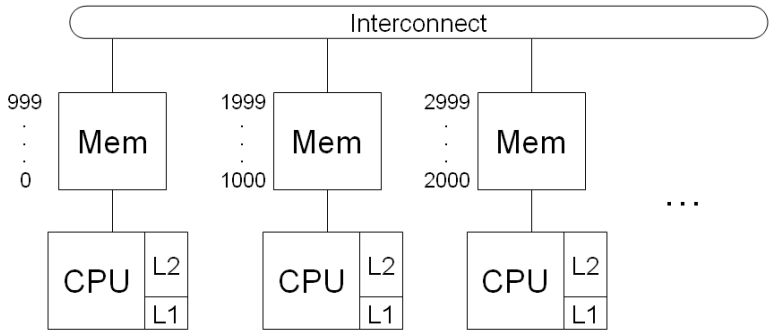


Figure 2.5: Distributed shared memory architecture. Memories are physically distributed but share a same address space.

share a single address space. Therefore, at any moment in time there can be only one copy of a memory block in the L2 cache. There are several advantages of shared cache, especially for on-chip multi-processor systems. First, the total cache capacity increases, which increases the cache hit ratio and reduces main memory bandwidth requirements. Also, cache coherence at the L2 level is not necessary, thus reducing the complexity of the system. The trade-off is that processors have to access blocks of memory that are stored in caches at remote locations, which increases cache hit time and on-chip bandwidth demands. Fortunately, the NoC design provides high on-chip bandwidth and low latencies, making the shared cache a viable alternative.

The NUCA is the dominant architecture in commercial multi-core systems. Because the number of cores in those architectures is usually small, most systems employ a single memory bank for all L2 (or L3) caches, similar to Figure 2.6. However, this design is not scalable because, as the number of processors increases, some processors will be located far away from the memory bank and will have a much higher cache latency than others. A more scalable solution is shown in Figure 2.7. This tiled CMP architecture has an L2 cache associated with each processor and an NoC is used for remote cache access and cache coherence. Either shared or private caches can be implemented in this architecture.

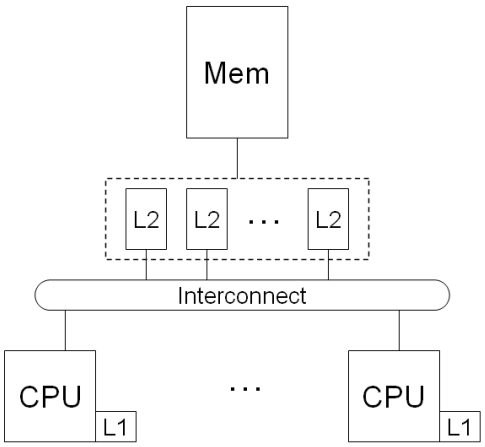


Figure 2.6: Non-uniform cache access (NUCA) architecture. This system has a centralized shared memory, but different from the UMA architecture, the level 2 caches are also shared.

2.2.2 Cache Coherence

In order to understand how communication occurs in shared memory systems, in this section we briefly review the cache coherence problem and its basic implementation protocols. Suppose the following sequence of events occur in a shared-memory computer:

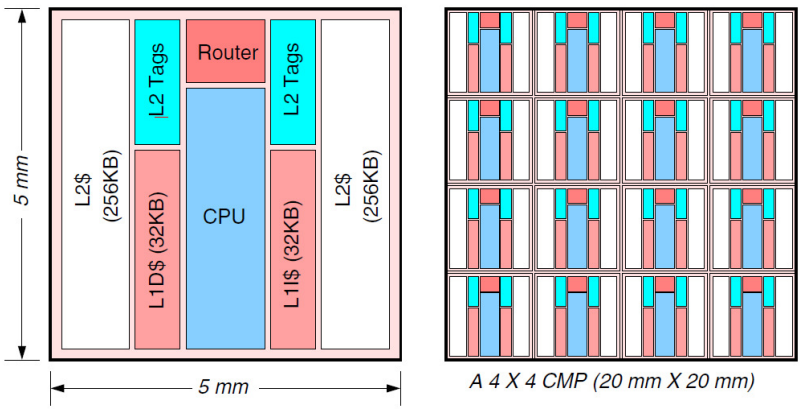


Figure 2.7: Tiled chip multi-processor architecture. (Figure reproduced from [97])

1. Multiple processors hold a same memory block in their private cache.
2. One of the processors writes on that memory block.
3. In order to ensure that all processors see this change, it is necessary that the other copies are updated with the new written information.

Based on the above, we can define cache coherence as follows: *cache coherence is the property that guarantees that all processors see the same value in a shared memory address*. The easiest and safest way to do this is to update all copies as soon as any processor writes on it. However, this leads to poor parallel processing performance because all processors have to stall until their copies are updated. As a result, relaxed models exist in which the copy update might be delayed. *Memory consistency is the property that guarantees that the correct result of the program will not be affected by delays in the memory updates*. Many consistency models exist, which define when the cache copies must be updated, such as sequential and processor consistency. The implementation of memory consistency is called synchronization. In shared memory systems, synchronization can be implemented implicitly using cache coherence.

There are two basic ways of implementing cache coherence, namely the snooping and the directory-based protocols. The snooping protocol has reduced latency, but requires a broadcast media and, therefore, is not scalable. The directory-based protocol is scalable to large number of processors, but it has increased latency and a slightly higher implementation overhead. Next, we give a brief introduction to both protocols.

Snooping Protocol

In this protocol, every cache that has a copy of the data from a block of physical memory also has a copy of the sharing status of the block, but no centralized state is

kept. To keep track of the state of a block stored in cache, three bits are associated to each copy:

- *Invalid*—States whether the copy in cache is valid (up to date) or not.
- *Modified*—States whether the copy has been modified. If true, the cache holds the unique updated copy.
- *Shared*—States whether the copy is shared by other processors. In this state, the copy is read only.

When a processor wants to access some data that is not in cache, it broadcasts a data request message to the other processors. If one of these processors has the data, it will send the data to the requesting processor. If no processor has the data in cache, the requesting processor fetches the value directly from memory. Figure 2.8 contains a diagram illustrating this situation for a write-back cache, with emphasis on the communication between processors. When a processor wants to write on shared data it must send an invalidate message to the other processors. It then acquires block ownership and is free to write on the block. A diagram showing the main steps is given in Figure 2.9.

Because of broadcasting, the snooping protocol consumes excessive bandwidth and is limited to a small number of processors, such as 2 to 4. A more scalable option for cache coherence is given by the directory-based protocol.

Directory-based Protocol

The directory-based protocol avoids broadcasting by keeping the shared status of a block in just one location, called the *directory node*. The directory keeps track of

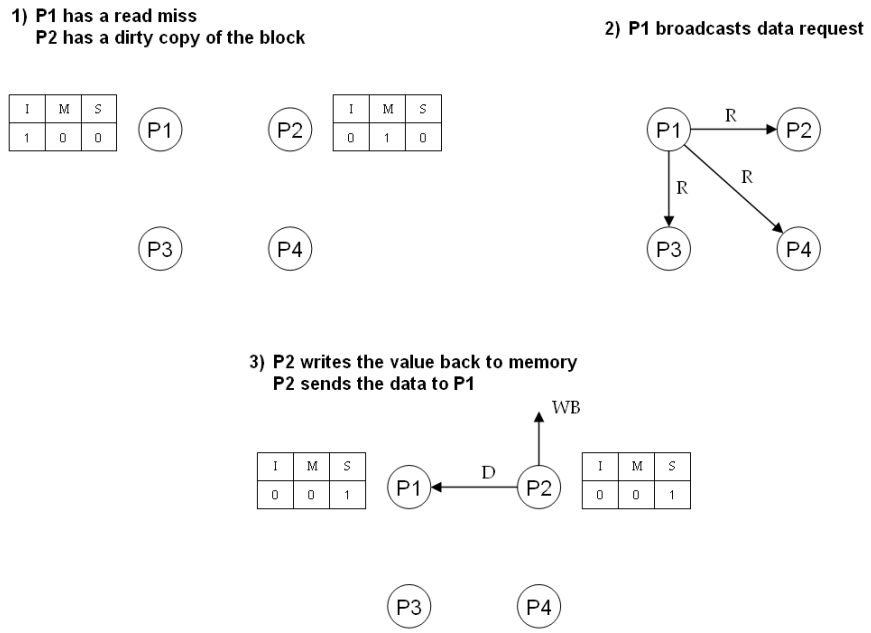


Figure 2.8: Basic steps of a snooping protocol when a processor requests data. The fields I, M, and S correspond to the invalidate, modified, and shared bits. R represents a data request, D data reply, and WB data write-back to memory.

all memory locations that might be cached. This protocol achieves scalability by making each processor the directory node of a different part of the physical memory.

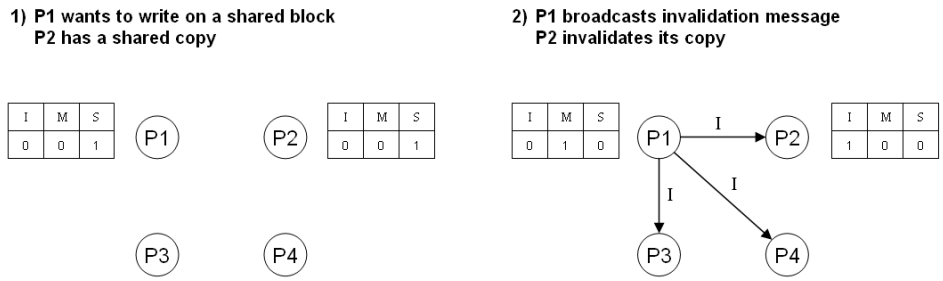


Figure 2.9: Basic steps of a snooping protocol when a processor wants to write on shared data. The fields I, M, and S correspond to the invalidate, modified, and shared bits. The arrows associated with a letter I are invalidate messages.

Similarly to the snooping protocol, whenever a processor wants to access data that is not in its cache it must request those data. However, instead of broadcasting a request message, it now sends a message exclusively to the directory node responsible for that block. The directory node then forwards the request to some other node that has a copy, or it might fetch the data from memory and forward it to the requesting processor if no processor has a cached copy. Note that the directory node must know whether the copy is cached and it also must keep track of what processors have a shared copy. To accomplish this, two new fields are necessary in the directory:

- *Uncached*—A bit stating whether a block of memory has no copies in cache.
- *Sharers*—A bit vector with all processors that share a cached copy.

Figure 2.10 shows the basic steps of a data request in the directory protocol.

When a processor needs to write on shared data, it must request ownership to the directory node, which sends invalidate messages to all processors that share the data. The requesting node then acquires ownership to write on the data. This process is illustrated in Figure 2.11.

Note that the directory-based protocol always requires one additional step than the snooping protocol. However, this solution exchanges slightly increased latency for higher scalability. Many other events can occur in cache coherence that are not described above. There are also several implementation details that were not covered. A more thorough account of snooping and directory-based protocol is given in [39].

2.3 Mapping and Scheduling

When running a parallel application on multiple processors the compiler or the operating system need to decide *where* (in what cores) and *when* tasks should run. The

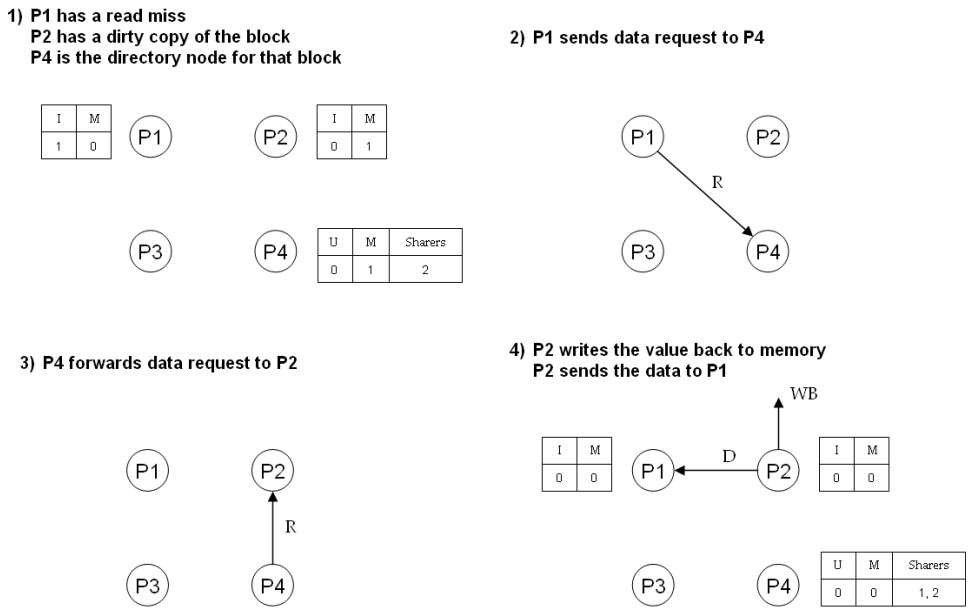


Figure 2.10: Basic steps of a directory-based protocol when a processor requests data. The fields I, M, and U correspond to the invalidate, modified, and uncached bits. R represents a data request, D data reply, and WB data write-back to memory.

former problem is called *application mapping* and the latter *application scheduling*. Oftentimes deciding *when* implies defining *where* the application should run and vice-versa, therefore it is not always the case that these two problems are independent of each other.

Most mapping and scheduling techniques rely on a graph-based description of applications called Task Graph (TG). The TG is a directed acyclic graph in which nodes correspond to tasks and edges correspond to control dependencies, meaning that a task can only start after the antecedent ones have completed. A task is any collection of operations that can be executed independently. Weights associated to edges indicate the CPU time required to conclude the execution of a task. Figure 2.12 depicts a typical example of a task graph.

The objective of mapping and scheduling is usually to minimize the execution

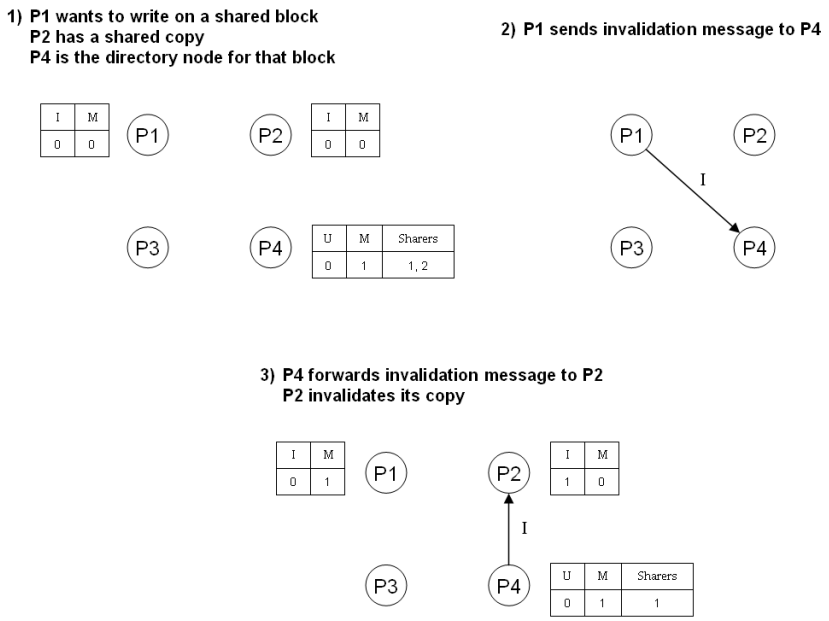


Figure 2.11: Basic steps of a directory-based protocol when a processor wants to write on shared data. The fields I, M, and U correspond to the invalidate, modified, and uncached bits. The arrows associated with a letter I are invalidate messages.

time of applications and/or energy consumption. These optimization objectives are closely tied to maximizing communication locality, since the shorter the communi-

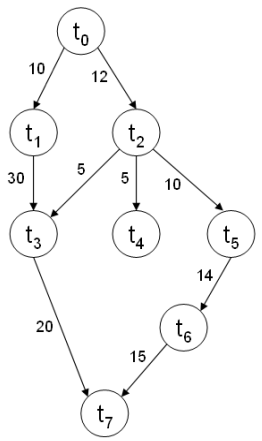


Figure 2.12: Task graph.

cation distances the less energy and latency involved in communication. Even in its simplest versions, mapping and scheduling are NP-Hard problems that require heuristics to be solved in practice. Lei and Kumar [62] proposed a genetic algorithm for mapping task graphs to an NoC, targeting execution time minimization. They assume a heterogeneous NoC, in which tasks have different execution times depending on the selected IP, and used synthetic TGs as experimental data. Hu and Marculescu [45] proposed a mapping and scheduling method to minimize energy consumption on a heterogeneous 2D mesh NoC, in which energy is modeled using the bit-energy approach. Their algorithm decides what tasks will be executed in each processing element and, if more than one task are assigned to the same processor, when each task will be executed. They used artificial data generated using Task Graphs For Free (TGFF) [28] and a set of benchmark multimedia applications.

Chen *et al.* [17] proposed a compile-based approach for mapping and scheduling applications on CMP. The approach works by first scheduling a TG on virtual processors targeting performance optimization and then by mapping the virtual processors onto physical processors. This second step occurs in a topology-aware manner in order to maximize locality. Locality is improved by making processors that share data to be placed closed to each other. They tested their method on the SpecFP2000 benchmark [40]. Kandemir and Chen [49] proposed an operating system-based scheduler for CMP which increases data locality in two ways. Firstly, the processes that do not share data are scheduled in different cores and, secondly, processes that could not be executed at the same time due to dependencies but share data are mapped to the same core. Their scheduler targets performance optimization and the data used consists of array-based image/video processing applications. Pop and Kumar [76] present a methodology based on GA to map and schedule applications for multi-threaded heterogeneous cores. They tested their technique on a 2×2 NoC using TGFF synthetic data. Saeidi *et al.* [80] presented a Matlab tool for mapping and scheduling applications on homogeneous NoC platforms. Task graph data are syn-

thetically generated and mapping is performed using a GA.

The works above employ static mapping and scheduling, which are performed at compile time. This approach has the advantage of reduced execution overhead, but might lead to poor performance in unpredictable environments and is architecture dependent. An alternative approach is dynamic scheduling, which is performed at execution time. Dynamic scheduling leads to higher execution overhead, but is architecture independent, and is more adequate for unpredictable applications and computing environments. Most dynamic techniques are based on sub-optimal, greedy solutions. State-of-art dynamic schedulers are work stealing (WS) and parallel depth first (PDF). WS works by maintaining a double ended work queue for each processor. When forking a new thread, the thread is placed on top of the local queue. When a processor looks for ready to execute tasks, it looks in its local queue first and takes a task off the queue if there is any. If the queue is empty, it checks the work queues of other processors and "steals" a task from the bottom of the queue. In PDF, when a core completes a task, it is assigned the ready to execute tasks that the sequential program would have executed the earliest. As a result, PDF tends to schedule tasks in a way that tracks in some sense the sequential execution. A comparison between WS and PDF shows that PDF provides better cache usage, thus optimizing performance relative to WS [18].

Chapter 3

Power Scaling in NoC Topologies

Interconnection networks of future multi- and many-core microprocessor systems will be required to deliver high performance at low power consumption to tens or even hundreds of cores [84]. In order to design energy efficient Networks on Chip (NoC) for systems of this magnitude it is necessary to understand how different network designs scale in terms of power and performance as the number of cores increases.

An important decision in NoC design is the network topology. Topological parameters, like hop count, bisection bandwidth, and wiring layout are closely related to the performance and power dissipation of a network, and may have considerable impact on scalability. NoC topologies have been studied extensively, e.g., [60, 74], and many alternative topological structures have been proposed [36, 51, 50, 66, 83]. However, few of these works focus on scaling.

In this chapter, we devise a theoretical model for power scaling in NoC topologies under uniform random traffic and use this model to analyze the trade offs between power and throughput in NoC. Uniform random traffic is a commonly used traffic pattern for NoC topology evaluation in which all nodes have an equal probability of talking to each other. In this study, we selected three topologies: binary tree, 2D

mesh, and fat tree, which differ considerably in their bisection bandwidth scaling—$O(1)$, $O(N^{1/2})$, and $O(N)$, respectively, where N is the number of cores—, thus covering a representative range of scaling behaviors. In order to verify our theoretical model, we performed computer simulations using Orion [90] to measure power and throughput of the network topologies as a function of the number of cores.

3.1 A Theoretical Model of Power Scaling

Dynamic power dissipation in NoC topologies has two components: power consumed on the routers and power consumed by driving the wires between routers. In the following analysis, we assume that power on each router is proportional to the flow of packets in the router, i.e., the number of packets the router processes per unit of time. For each wire, we assume that power is proportional to the flow of packets times the wire length, since for repeated wires power increases linearly with the number of repeated segments. Secondary effects, such as the switch control path and virtual channel allocation, were ignored here for the sake of simplicity. Total power in a given network is obtained by summing up over all routers and all wires. Finally, accuracy is improved by normalizing the router and wire components of total power by the flow on the busiest router and busiest wire, respectively, as those components represent potential bottlenecks to the performance of the topology and constrain the maximum power of the whole network.

As an example, consider the binary tree with 16 processor nodes shown in Figure 3.1. Each processor injects one unit of flow of packets in the network, which will be equally divided between the other 15 nodes, assuming uniform random traffic. The figure illustrates how the flow generated by one processor is distributed over the entire tree. The total flow on the network is obtained by summing the flow injected by all 16 processors. Notice the routing algorithm must be considered when

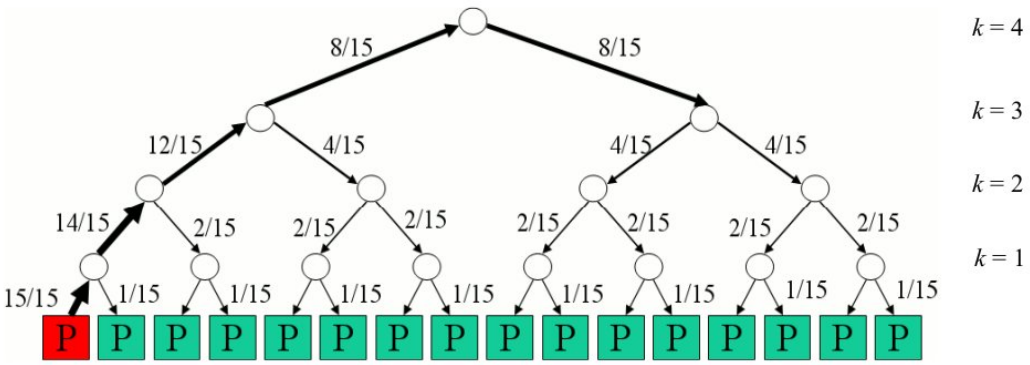


Figure 3.1: Flow of packets injected by one processor in a binary tree network with 16 processor nodes. The numbers represent the fraction of the flow passing through each path of the tree. The values of k represent the levels of the tree.

computing this flow. For binary and fat trees, nearest-common-ancestor routing was used.

According to the figure, a given processor sends all its packets to a router at the first level of the tree ($k = 1$) to which it is directly connected. All the other routers at the same level receive 2 units of flow from this processor. Thus, each router at level 1 receives $2 \times \frac{15}{15}$ units of flow coming from the 2 processors to which it is directly connected, and $14 \times \frac{2}{15}$ units of flow from the remaining 14 processors. Since there are 8 routers at this level, total flow at level 1 of the tree is $8 \times \left(\frac{30}{15} + \frac{28}{15}\right) = \frac{456}{15}$. Applying the same reasoning to all the levels of the tree and generalizing for an arbitrary number of processors, the following equation is obtained for the total flow on the routers,

$$Flow_{\,routers}(N) = \frac{N}{N-1} \sum_{k=1}^{\log N} \left[(N - 2^k) + (N - 2^{k-1}) \right], \tag{3.1}$$

where N is the number of processors. Power is obtained by normalizing equation 1 by the flow on the top router, which after some simplification yields:

$$Routers_{\,binary\ tree}(N) = \frac{c_r}{2N - 3 \cdot 2^{\log N - 1}} \left[2N \log N - 3 \cdot \sum_{k=1}^{\log N} 2^{k-1} \right], \tag{3.2}$$

where c_r is a constant that defines the power per packet in a router. For wires, the equation of flow will be similar to that of routers, but in order to calculate power it is necessary to weight the flow by the wire length at each level. For binary and fat trees, the H-tree layout was adopted, thus the length of the wires doubles at each hierarchical level. Since the die size is assumed to be constant as the number of processors increases, the length of the shortest wires decreases as N. Consequently, the length $l(k)$ of a wire at level k becomes,

$$l(k) = \frac{l_0 2^k}{\sqrt{N}}, \tag{3.3}$$

where l_0 is a constant denoting the length of shortest wire segment. The final equation for power on wires in a binary tree is:

$$Wires_{\,binary\,tree} = \frac{c_w}{\sqrt{N}(N - 2^{\log N - 1})} \sum_{k=0}^{\log N - 1} (N - 2^k)2^k, \tag{3.4}$$

where c_w is a constant that defines power per packet per wire segment. The same method when applied to mesh leads to the following equations:

$$Routers_{mesh} = \frac{c_w \cdot 8}{N-1} \sum_{x=1}^{\sqrt{N}-1} \left[x \left(N - x\sqrt{N} \right) \right], \tag{3.5}$$

$$Wires_{\,mesh} = \frac{c_w \cdot 8}{N^{3/2}} \sum_{x=1}^{\sqrt{N}-1} \left[x \left(N - x\sqrt{N} \right) \right], \tag{3.6}$$

where the variable x represents the x-coordinates for one side of a squared mesh. For the mesh topology, the xy-routing algorithm was used in the computation of flows. Finally, for the fat tree topology the power equations are given as:

$$Routers_{\,fat\,tree}(N) = \frac{c_r \cdot N}{N-2} \sum_{k=1}^{\log N} \left(N - 2^k \right), \tag{3.7}$$

$$Routers_{\,fat\,tree}(N) = \frac{c_r \cdot N}{\sqrt{N}\,(N-2)} \sum_{k=0}^{\log N - 1} \left(N - 2^k \right) 2^k, \tag{3.8}$$

Table 3.1: Closed-form solutions for power on different topologies.

Topology	Power on Routers	Power on Wires
Binary tree	$c_r \cdot 4 \log N - 6 - \frac{6}{N}$	$\frac{c_w \cdot 2}{N^{3/2}} \left(N^2 - 2N - \frac{4^{\log N}}{3} - \frac{4}{3} \right)$
Mesh	$c_r \cdot 4N$	$\frac{c_w \cdot 4(N-1)}{3\sqrt{N}}$
Fat tree	$\frac{c_r \cdot N}{N-2} (N \log N - 2N + 2)$	$\frac{c_w \cdot \sqrt{N}}{N-2} \left(N^2 - 2N - \frac{4^{\log N}}{3} - \frac{4}{3} \right)$

Table 3.2: Theoretical estimates for scaling in different topologies according to asymptotic analysis.

Topology	Total Power	Throughput	Energy per packet
Binary tree	$O(N^{1/2})$	$O(1)$	$O(N^{1/2})$
Mesh	$O(N)$	$O(N^{1/2})$	$O(N^{1/2})$
Fat tree	$O(N^{3/2})$	$O(N)$	$O(N^{1/2})$

3.2 Analysis of the Scaling Behavior

Here we give closed form solutions, shown in Table 3.1. Next, we characterize the asymptotic behavior of these topologies, observing that, for large enough values of N, the highest order terms dominate. For instance, in binary tree, power on the routers scales as $O(\log N)$ and power on the wires as $O(N^{1/2})$, thus the wire component dominates total power dissipation. For a 2D mesh, power is dominated by the routers as $O(N)$, and in the fat tree by the wires as $O(N^{3/2})$. Finally, by dividing power by throughput we measure energy per packet. Under uniform random traffic, throughput is expected to scale as the bisection bandwidth. Table 3.2 summarizes the expected scaling behavior of power, throughput, and energy per packet for all three topologies resulting from asymptotic analysis.

The values in Table 3.2 show that, although the topologies analyzed vary widely in

terms of power dissipation and throughput, energy per packet for all three networks scales in the same way. This result is somewhat surprising, given the emphasis in the literature about which NoC topologies are the most energy efficient. Notice that, since energy per packet is the same, the resulting power consumption in each topology is proportional to throughput times the average energy per packet. Thus, there is a linear trade-off between power and throughput, and topologies able to achieve high performance will, consequently, have high power consumption. This theoretical finding formalizes experimental results reported in the literature [74].

Therefore, for uniform random traffic and large enough N, the above analysis suggests that topology has negligible influence on the scaling of packet energy consumption.

3.3 Experimental Results

3.3.1 Simulation Infrastructure

All simulations were performed using Orion 1.0. For each topology, networks with 8 up to 1024 processor nodes were simulated and average power and throughput were measured as a function of the number of cores. Orion calculates router power into three separate components: memory, arbiter, and crossbar power. Links are automatically divided into repeated wire segments, based on length. Traffic injection rates were set high enough in each run to saturate the network. As a consequence, actual packet injection is proportional to maximum throughput, and may vary with both the topology and the traffic pattern.

All simulations were run in a $0.1\mu m$ process, which is the standard for Orion 1.0. Packets are comprised of 4 flits of 64 bits. Routers have 2 to 4 physical ports, 4 virtual channels, and a 12-flit output buffer. Each link is composed of 2 unidirectional, 64-bit

channels. A credit policy was used to account for buffer availability on routers. The routing algorithm used for the mesh topology was xy-routing, and for the binary and fat trees we used nearest-common-ancestor routing with randomized upward paths.

3.3.2 Results

In the following experiments, uniform random traffic is simulated and the results are compared with the theoretical predictions of the model presented in Section 3.2. Figure 3.2 shows the curves obtained for the three topologies. The confidence intervals are too small to be visible in the graphs. However, there is an excellent agreement between the theoretical model and the simulated results; the computed R^2 correlation coefficients were above 0.99 for all the curves.

The scaling behavior of throughput and power varies for each topology. For instance, fat tree achieves the highest throughput, with linear increase in the number of delivered packets per cycle as N increases. It also has the highest power consumption, since both, power on routers and power on wires scale super-linearly with the number of cores, as $O(N \log N)$ and $O(N^{3/2})$, respectively. In binary tree, the scaling of power consumption is relatively small: power on routers grows as $O(\log N)$ and on wires as $O(N^{1/2})$, a slower than linear rate of increase in both cases. However, its throughput remains constant and does not scale with the number of cores. Mesh represents an intermediate case, with moderate throughput and power scaling. For this topology, power on routers scales approximately linearly with N, and faster than power on wires, which increases as $O(N^{1/2})$.

In contrast to power and throughput, the scaling curves of energy efficiency for the three topologies are remarkably similar. This result agrees with the theoretical predictions of the asymptotic analysis in Section 3.2, and shows that topological properties, such as bisection bandwidth, number of routers and wire layout, which

have high impact on throughput and power, have little influence on the overall energy efficiency of the networks.

Although they are similar, the energy efficiency curves are not identical. Some discrepancy is expected because, as predicted by the theory, scaling in the three topologies is the same only for large values of N. Also, implementation details of the simulations may affect the scaling constants of each topology in a slightly different way. For example, because in the mesh topology, power on the routers scales faster than power on the wires, using a smaller constant for power on the routers will favor mesh over binary and fat tree in the scaling of energy per packet.

The experimental results show that, for uniform random traffic, energy per packet increases significantly as the number of cores increases and sending a message over the network becomes increasingly expensive. This happens because uniform random traffic lacks any locality of communication, leading to poor NoC scalability.

3.4 Conclusion

In order to design scalable multi-core architectures with thousands of cores, it is important to understand how power consumption on the NoC increases as the number of cores increases. In this chapter, we presented a theoretical model for the scaling of power and performance of different NoC topologies under uniform random traffic. Our main result is that there is a linear trade-off between throughput and power, i.e., topologies that deliver higher performance have higher power consumption and vice-versa, in a linear fashion. As a result, energy efficiency scales in the same way independent of the topology. In this scenario, locality of communication and not topology is the main variable affecting energy efficiency, as will be seen in subsequent chapters. Our models showed excellent agreement with simulation results, which validates the accuracy of our theoretical approach.

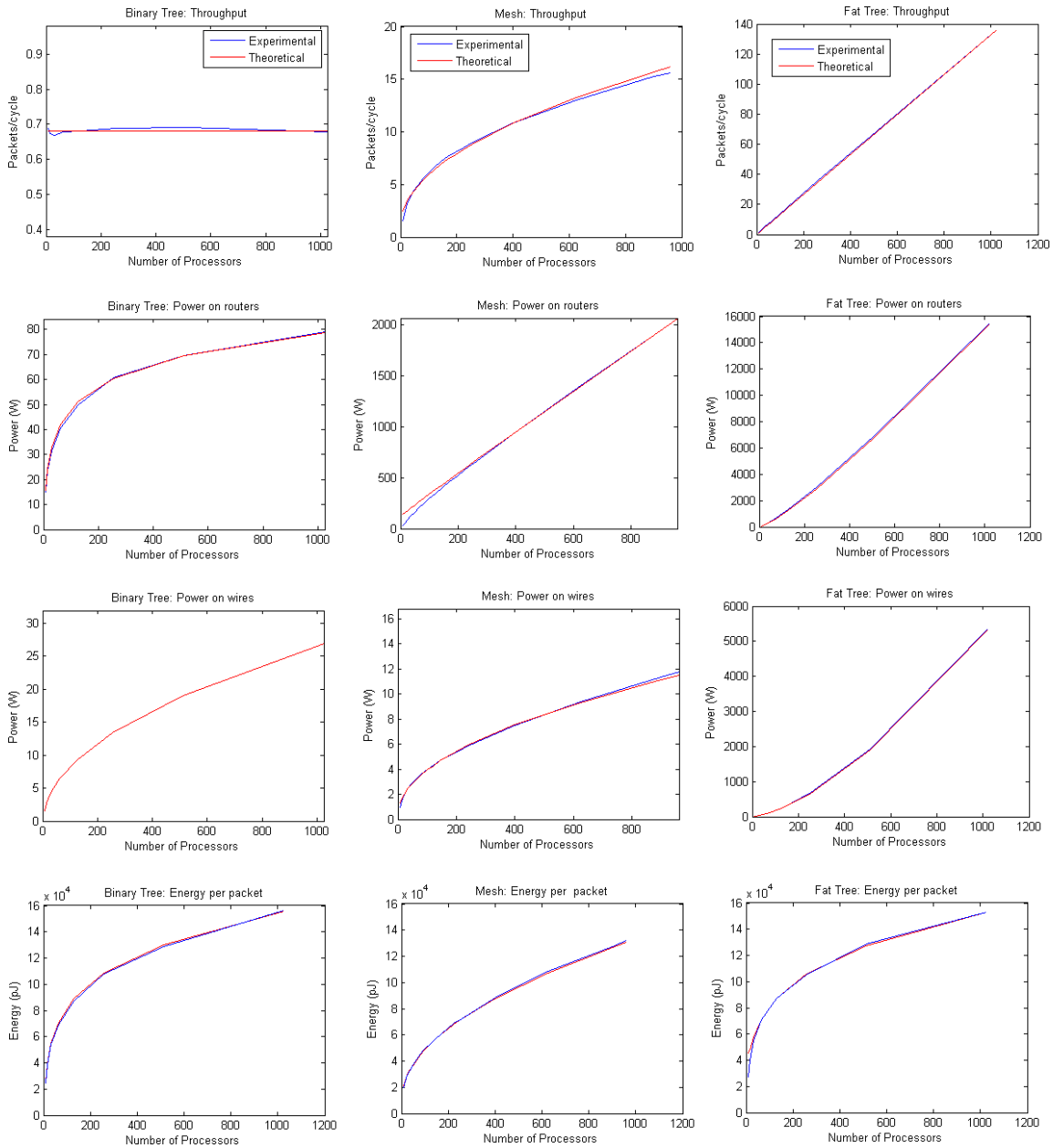


Figure 3.2: Comparison between the theoretical results of our models with simulation results

Chapter 4

Modeling NoC Communication Locality using Rent's Rule

Rent's rule is an empirically observed pattern in VLSI designs that describes the communication structure between logic gates on a chip. Using derivations based on Rent's rule, the Wire Length Distribution (WLD) of a circuit can be estimated from its Rent's exponent and coefficient, p and k [25]. This distribution describes the communication locality in the circuit and, therefore, is related to many properties of the system, such as chip area, signal delay, power consumption, and wire routability [86].

In Systems on Chip (SoC), similar information is provided by the Communication Probability Distribution (CPD) of applications. The CPD describes the probability that packets will travel a certain distance in the Network on Chip (NoC) for a given traffic pattern. This distribution is directly related to the energy consumption of an application, because the larger the distance traveled by packets, the more energy is used. Since current NoCs use 30 to 40% of the power budget [88, 43], it is desirable for the distance traveled by packets to be as small as possible in order to minimize

this cost.

In this chapter, we use the CPD to study NoC traffic locality and energy consumption. Firstly, motivated by the importance of Rent's rule to VLSI and supported by recent work showing that communication patterns of many parallel applications follow Rent's rule [38], we propose a method for generating Rent's rule traffic patterns. In this method, the probability of communication between processors is derived directly from Rent's rule, leading to CPDs displaying high traffic locality. This method could be used to simulate traffic as a fast and simple alternative to application-driven workloads.

Based on the CPD, we also propose a model for predicting energy consumption in a network on chip. We tested the model on several synthetic workloads, including Rent's rule traffic, running on two different NoC systems and compared the obtained results with architecture-level simulations. The results show excellent agreement between predicted and experimental values. Our approach does not require simulation and could be used in the early phases of NoC design, and it could aid the design of energy-efficient applications and better application mapping techniques [44]. Finally, using our traffic generator we also analyze the impact of the Rent's exponent of an application on energy consumption.

4.1 Rent's Rule Traffic patterns

4.1.1 Rent's Rule for Parallel Programs

In VLSI, Rent's rule emerges naturally from circuit placement, in which connections are made as local as possible to minimize wire footprint, power and latency [22]. Similar constraints apply to the communication among processors in multi- and many-core systems. Algorithms used for mapping parallel applications onto cores

aim at producing optimized layouts that minimize communication distances.

Greenfield *et al.* [35] argue that, analogous to circuit placement in VLSI, Rent's rule will naturally arise in multi- and many-core chips from this optimization process. They extended the concept of connection locality in circuits to communication locality among cores, proposing a bandwidth-based version of Rent's rule,

$$B = bN^p, \tag{4.1}$$

where B is the bandwidth sent or received by a cluster of N network nodes, b is the average bandwidth per node, and $0 \leq p \leq 1$ is the Rent's exponent.

In recent work, Heirman *et al.* [38] showed that many parallel applications indeed follow Rent's rule. They analyzed 13 popular benchmark applications running on 32 and 64 cores. Using a partitioning algorithm they showed that all of the programs followed Rent's rule with measured values of the Rent's exponent p ranging from 0.55 to 0.74.

4.1.2 Generating Rent's Rule Traffic Patterns

The discussion above motivates the use of a synthetic generator of traffic that follows Rent's rule. Such a traffic generator could serve as a simple way to evaluate NoCs with workloads that mimic the spatial properties of real traffic. As will be discussed in Section 4.2, many existing synthetic workloads correspond to special case situations used to stress the network and routing algorithm. However, the authors are unaware of work that employs Rent's rule synthetic traffic as a generic model of communication in parallel applications.

In VLSI, the probability of a wire connecting two terminals with Manhattan distance d apart is given by (adapted from [25]):

$$P(d) = \frac{(1 + d(d-1))^p - (d(d-1))^p + (d(d+1))^p - (1 + d(d+1))^p}{4d} \tag{4.2}$$

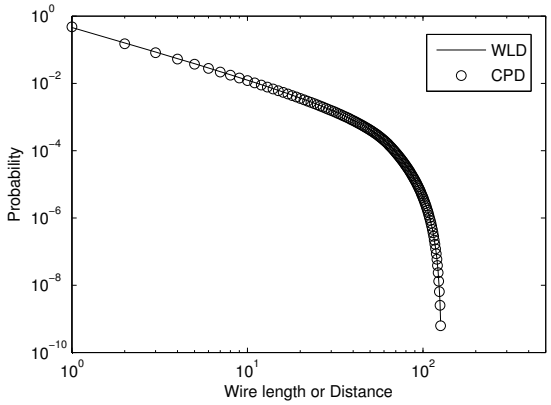


Figure 4.1: Comparison between the wire length distribution given by [25] and the communication probability distribution produced by the Rent's rule traffic generator.

We use the equation above to define the probability of communication between two processors, where d corresponds to the number of hops in the shortest path between source and destination. Traffic can be generated for each source node by sampling from the probability in Equation 4.2 for every possible destination node. Repeating this process for all possible source nodes in the network results in traffic that follows Rent's rule.

To validate our method, we generated traffic using equation 4.2 and measured the resulting CPD. This distribution was then compared to the wire length distribution given by Davis *et al.* [25], which is derived directly from Rent's rule and is widely used in wire length estimates of real circuits. Figure 4.1 shows a log-log plot of the comparison between the wire length distribution given by [25] and the CPD produced by our traffic generator. The plot shows a virtually exact match between the two curves. In this figure, $p = 0.75$, which is a typical exponent for VLSI architectures, and the network has 1024 nodes.

The formula for the CPD of synthetic Rent's rule traffic can be derived from

Equation 4.2 and is given by:

$$CPD(d) = \Gamma P(d) \cdot \sum_{i=1}^{2\sqrt{N}-2} \left(\sqrt{N} - i\right)\left(\sqrt{N} + i - d\right), \tag{4.3}$$

$$\text{for } 0 < \left(\sqrt{N} + i - d\right) \leq \sqrt{N}.$$

where Γ is the normalization coefficient such that

$$\sum_{d=1}^{2\sqrt{N}-2} CPD(d) = 1.$$

Figure 4.2(a) shows the CPD produced by our generator on an 8×8 mesh network.

Another interesting property of this method is the ability to generate traffic patterns with arbitrary Rent's exponents. Because the Rent's exponent is related to communication locality and complexity of applications, it is possible to study the NoC under several application scenarios by varying a single parameter in the model.

4.2 Other Synthetic Workloads

In this section, we review some commonly used synthetic traffic patterns and compute their CPD, which is similar to the spatial hop distribution presented in [85]. We compare the obtained distributions with the CPD of Rent's rule traffic.

Uniform Random Traffic In uniform random traffic, each source is equally likely to send packets to each destination. This is the most commonly used traffic pattern for network evaluation because it is straightforward to implement, it makes no assumptions about the application, and it is analytically tractable. Because source nodes do not differentiate between near and distal destination nodes, uniform random traffic does not exploit locality of communication. Figure 4.2(b) shows the CPD for uniform random traffic on a 8×8 mesh network.

Bit Permutation Traffic In permutation traffic, each source *src* sends all of its traffic to a single destination, $des = \pi(src)$, where π corresponds to a permutation function. Because this type of traffic concentrates load on individual source-destination pairs, they tend to stress the load balance of a topology and routing algorithm. Bit permutations are a subclass of permutations in which the destination address is computed by permuting the bits of the source address. The CPDs of bit transpose, bit complement and bit rotation permutation traffic are show in Figure 4.2(c), 4.2(d) and 4.2(e), respectively. These distributions are considerably different from each other as well as from uniform random traffic. Details on how to generate these traffic patterns are given in [23].

Nearest Neighbor Traffic Nearest neighbor traffic is commonly used to evaluate the impact of communication locality on the performance and power consumption of the network on chip [73]. A fixed percentage of traffic goes to the nearest neighbors with some radius r and the rest of the traffic is uniform and random. The CPD of nearest neighbor traffic with $r = 1$ and locality factor of 50% is shown in Figure 4.2(f).

The traffic patterns described above are useful in practice as special cases to analyze the network, but bear little or no resemblance to real traffic. When compared to Rent's rule traffic (Figure 4.2(a)), most of these workloads display poor communication locality. As will be seen in Section 4.4, these differences in the CPD have considerable effect on the energy consumption of the NoC.

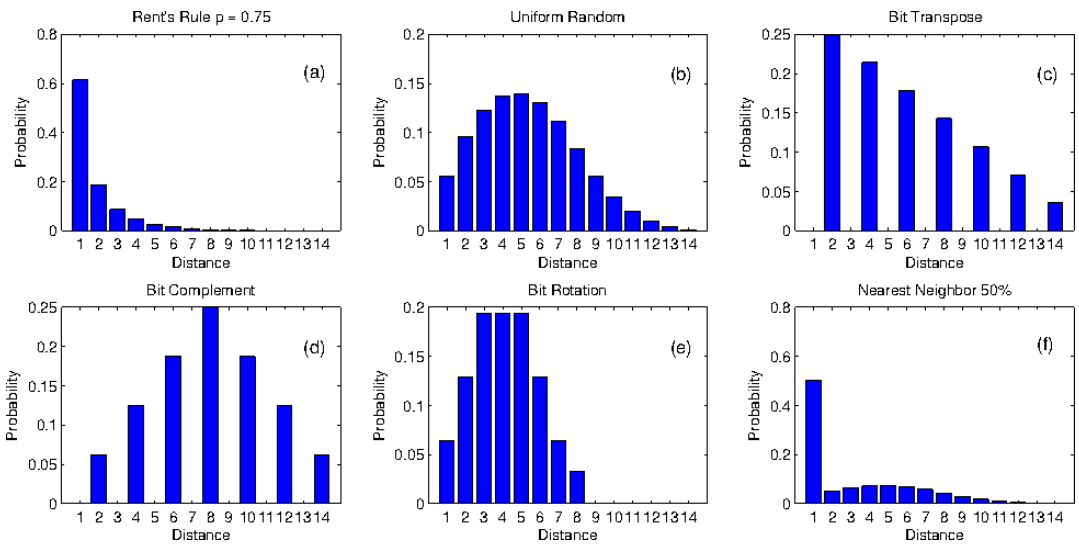


Figure 4.2: CPD of different traffic patterns on a 8×8 mesh network. (a) Rent's rule with Rent's exponent of 0.75.(b) Uniform random. (c) Bit transpose. (d) Bit complement. (e) Bit rotation. (f) Nearest neighbor with localization factor of 50%.

4.3 Modeling Energy Consumption

It can be computationally expensive to analyze NoC energy consumption using simulations, especially with application-driven workloads or large system sizes. In this section, we provide a simple model for predicting energy consumption based on the CPD, which does not require computer simulations. This model is intended for direct networks in which the length of the wires is the same for every hop, such as mesh and folded torus, but it could be easily extended to other topologies.

The average energy of a flit traversing a path of length d in the network is given by

$$E_{flit}(d) = d \cdot E_{link} + (d+1) \cdot E_{router}, \tag{4.4}$$

where E_{link} and E_{router} are the energy consumed by the flit when traversing a link and a router, respectively, and d is given by the number of hops traversed in the path. The total energy consumed by an application is obtained by first summing E_{flit} over

all communication distances weighted by the probability of a packet traveling that distance. This value is then multiplied by the number of flits per packet (N_{flits}) and the total number of packets ($N_{packets}$):

$$E_{total} = N_{packets} \cdot N_{flits} \cdot \sum_{d=1}^{max} E_{flit}(d) \cdot CPD(d). \tag{4.5}$$

In Equation 4.5, we assume a constant number of flits per packet. The constants E_{link} and E_{router} used in Equation 4.4 can be obtained from architecture-level power models, such as Orion 2 [48].

For traffic that follows Rent's rule, the model presented above provides a unique advantage over other approaches [44, 72, 73]. Given the Rent's exponent, the CPD of traffic can be directly obtained from Equation 4.3. With this information, the energy consumption of an application can be easily predicted from Equation 4.5. Our model's ability to predict energy usage for Rentian traffic based on a single application parameter could significantly simplify and speedup NoC energy analysis.

A potential limitation of this method is the assumption that the energy used for communication is proportional to the distance traveled by packets. This is approximately true for most networks on chip and is commonly used in the literature as a simplification step [44, 72, 73]. However, contention in the network could lead to extra dynamic and static energy that are not accounted for by the model.

4.4 Experimental Results

4.4.1 NoC Energy Consumption

We analyzed the energy consumption of different traffic patterns and tested the predictions of Equation 4.5 on two NoC configurations with different process technologies. The first system is an 8×8 mesh network running at 1GHz, on a 1×1cm

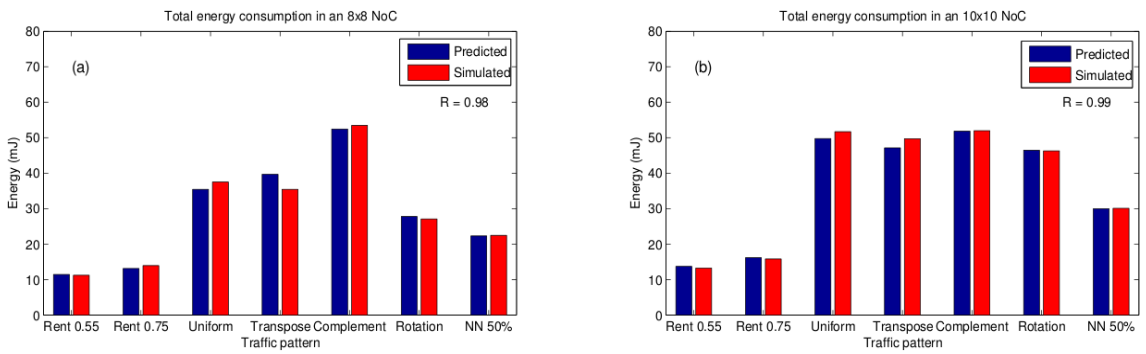


Figure 4.3: Predicted and simulated energy consumption for (a) 8×8 mesh NoC on 65nm and (b) 10×10 mesh NoC on 45nm.

die, and 65nm technology. Flit size was set to 64 bits and packets have five flits each. The routing algorithm was dimension-order routing with wormhole flow control and 4 virtual channels. Constants for flit energy were obtained using Orion 2 assuming activity factor of 0.5. For each of the traffic patterns, 20,000 packets were injected in the network. The exponents used for Rent's rule traffic were $p = 0.55$ and $p = 0.75$, corresponding to the two extremes of Rent's exponents measured in [38]

The energy predictions were compared to computer simulations and the obtained values are shown in Figure 4.3(a). The results show excellent agreement between predicted and experimental energy values, with correlation coefficient of 0.98. Table 4.1 shows the same results in more detail. The best prediction was obtained for nearest neighbor traffic, with 0.7% error, and the worst for bit transpose, with error of 12.01%. As discussed in section 4.3, prediction errors can be explained by nonlinear factors in energy consumption and differences in network contention for each traffic pattern.

The second system is a 10×10 network, on 45nm process technology and clock frequency of 3GHz. Flits have 32 bits each and the packet size is ten flits. The results are shown in figure 4.3(b). For this system, there is also a close match between predicted and experimental values, with correlation coefficient of 0.99. The

Table 4.1: Predicted and simulated energy values for 8×8 and 10×10 NoCs. The uncertainty values arise from the limited number of packets sampled from the CPD.

	8×8 NoC			10×10 NoC		
Traffic	Pred.(mJ)	Sim.(mJ)	% Err.	Pred.(mJ)	Sim.(mJ)	% Err.
Rent 0.55	11.43	11.21±0.09	+2.00	13.69	13.25±0.03	+3.32
Rent 0.75	13.11	13.92±0.08	–5.78	16.15	15.79±0.03	+2.26
Uniform	35.44	37.51±0.15	–5.51	49.76	51.70±0.13	–3.74
Transpose	39.69	35.43±0.11	+12.01	49.18	49.73±0.10	–1.10
Complem.	52.43	53.46±0.13	–1.94	51.84	51.97±0.13	–0.23
Rotation	27.77	27.08±0.04	+2.52	47.21	46.29±0.09	+1.97
Nearest N.	22.30	22.46±0.08	–0.70	29.96	30.09±0.05	–0.44

results are shown in detail in table 4.1. A maximum error of 3.74% was obtained for uniform random traffic and a minimum error of 0.23% for bit complement.

The results above show that the proposed model produces accurate results over a wide range of traffic patterns, for different system configurations and also across different technology generations. This methodology could be used as a simple and fast tool for first-order assessment of energy consumption once the communication pattern of an application is known. Figure 4.3 also shows that Rent's rule traffic consumes the least energy when compared to the other workloads, especially for the 10×10 system. This could be predicted from the CPDs in figure 4.2, since this is the traffic with the most communication locality. Because it is based on empirical data, it should be expected that Rent's rule traffic provides a better model of communication locality of real applications than the other synthetic workloads.

4.4.2 Varying the Rent's Exponent

For VLSI devices, the value of the Rent's exponent is commonly used as a measure of circuit complexity. Simple, highly regular circuits have small values of the Rent's exponent, which are associated with high locality of communication. Conversely,

the Rent's exponent is large for more complex circuits in which a significant part of the communication is global. Analogously, in the bandwidth version of Rent's rule, small values of p represent simple applications with mostly nearest-neighbor communication, while large values correspond to applications with relatively poor communication locality. In this section, we analyze the impact of the Rent's exponent on the energy used for communication, which could have important implications to application design.

We generated Rent's rule traffic for a variety of Rent's exponents and measured the energy consumption for three network sizes: 6×6, 8×8, and 10×10. The process technology used in the simulations was 45nm for all three systems. The results depicted in figure 4.4 show a significant increase in the energy consumption as the Rent's exponent increases in all three networks. The impact of the Rent's exponent on energy is also stronger for the larger systems. As p varies from 0.1 to 0.9, there is an increase of 51% in energy for the 6×6 NoC, 68% for the 8×8 NoC and 83% for the 10×10 network.

These results show quantitatively that the price to be paid for communication complexity is high and will tend to increase in the future. As we move towards larger systems with potentially hundreds of cores, the demand for less complex and more energy-efficient applications will increase. Energy-efficient algorithms are an important topic in other fields, such as sensor networks [16], and will likely become a major issue in application design for systems on chip.

These experiments illustrate the flexibility of our synthetic traffic generator and its applicability in the analysis of NoC. By varying the Rent's exponent, it is possible to generate a continuum of application complexity scenarios, even ones that do not exist yet, and for systems with arbitrary sizes. The analysis presented here would not be possible with conventional execution-driven and trace-driven application workloads, which are limited to existing applications only.

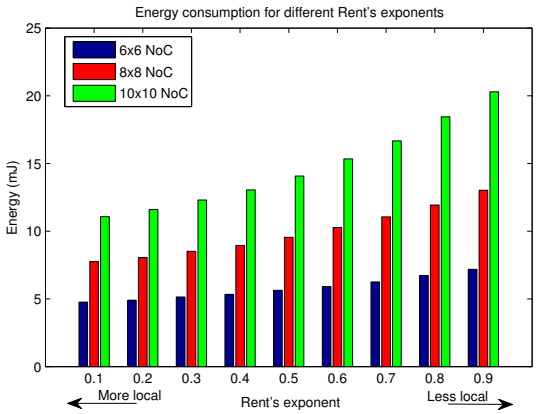


Figure 4.4: Energy consumption of 6×6, 8×8, and 10×10 NoCs for Rent's rule traffic as a function of the Rent's exponent.

4.5 Discussion and Conclusion

In this paper we used the CPD to model traffic locality and energy consumption in NoC. We proposed a synthetic traffic generator based on Rent's rule that mimics the CPD of traffic patterns for real applications. This method can be used as simple way to evaluate NoC designs under a variety of application complexity scenarios without having to resorting to application-driven workloads.

Although the method is designed to be more realistic than commonly used synthetic traffic patterns, it has some limitations. For example, temporal aspects such as burstiness and variations of the Rent's exponent over time [38] were not considered. Also, many applications exhibit traffic patterns with a central node, which might be better modeled with a combination of Rentian and hotspot traffic. Extending the model to consider these factors is a promising direction for future work.

Based on the CPD, we also proposed a simple model for predicting NoC energy consumption. The model is based on the assumption that energy is proportional to the distance traveled by packets. We tested our model on two system configurations and 6 different traffic patterns, with accurate results. One advantage of this model

is the ability to predict energy directly from the Rent's exponent for traffic patterns that follow Rent's rule. The results also showed that the energy consumed by Rent's rule traffic is less than that of other synthetic workloads, because it has more locality of communication.

Finally, we used Rent's rule traffic patterns to analyze the impact of the Rent's exponent on NoC energy consumption. We showed that the cost of communication complexity is significant and will likely become a constraint on the scalability of future NoCs.

Chapter 5

Data Placement Optimization for Chip Multi-Processors

In this chapter, we present a new method for data placement optimization in CMPs, which manages the trade-off between communication locality and load-balancing to reduce the energy consumption on the interconnect. Assuming the communication graph of an application is known, our method reduces energy by minimizing communication distances while increasing cache capacity utilization, which reduces unnecessary network traffic. Simulations on a 64-core system show a reduction in the NoC dynamic energy consumption of 49.8% on average and as high as 84.1%, with performance gains of up 16.9% on shared-memory implementations of the SPLASH-2 benchmark. These results outperform those obtained with greedy and first-touch placement strategies. Unlike heuristic methods with no guarantees on the quality of the solutions, our algorithm is exact and can be solved in polynomial time.

5.1 Related Work

This section briefly describes the related work on data placement in CMPs and summarizes its limitations, which are addressed by our method. Many of these methods are adaptations of data placement policies for NUMA (Non-Uniform Memory Access) systems.

Several data placement strategies are based on alternative cache management polices that combine private and shared cache schemes. For example, reference [97] presents a victim replication cache management policy, which reduces cache hit latency by keeping copies of local data within the local L2 cache, while allowing for replication of shared data. A NUCA (Non-Uniform Cache Access) organization in which caches are completely unshared, partially shared, or completely shared was proposed in [46], and it was concluded that the ideal level of sharing depends on the application. In [67], a method is described that also attempts to reduce hit latency by partitioning the cache into private and shared content. Reference [20] proposed controlled replication for fast read-only sharing, *in situ* communication to restrict cache misses, and neighbors' capacity stealing when private data exceeds a core's capacity.

Other techniques have been proposed that are based on page or cache block migration. In [15], the pageNUCA policy is presented, which consists of a coarse-grain data migration mechanism that dynamically monitors the access patterns of cores to decide when to migrate a page. In [2], a page placement method is presented in which hardware and the OS dynamically manage cache capacity per thread and migrate shared data to improve locality. The approach proposed in [37] combines both migration and replication mechanisms. Their method is designed to react to different classes of applications to decide the appropriate location of blocks.

In [19], a distance associativity cache organization is proposed, in which place-

ment of data at a certain distance is separated from set associativity. A proximity-aware coherence mechanism is presented in [12], which accelerates read and write misses by initiating cache-to-cache transfers from the spatially closest sharer. This eliminates unnecessary accesses to off-chip memory and minimizes communication distances. Reference [21] proposes an OS-based cache management policy. By assigning blocks to caches at the page granularity, the OS can be used to implement arbitrary cache management strategies on demand.

Our approach differentiates from the works cited above in the following main aspects:

- All the above methods are heuristics which provide no guarantees on the quality of the solutions. Our method is exact and can be solved in polynomial time.
- Our approach focuses explicitly on energy consumption minimization. Of the above papers, only [19] reports improvements in energy consumption.
- Most of the works cited above employ generic solutions that treat every application in the same way. Our method produces mappings that are fine-tuned for individual applications and, therefore, can achieve higher improvements.

5.2 The Data Placement Problem

In shared-memory chip multi-processors, each core is the *home node* (or *directory node*) of a subset of the memory addresses and is responsible for managing the information and/or hosting in cache the memory blocks corresponding to those addresses. Memory accesses and cache-coherence operations require frequent communication with the home node. Therefore, the location of the home node of a block relative to the cores that frequently access it is highly important to the performance and

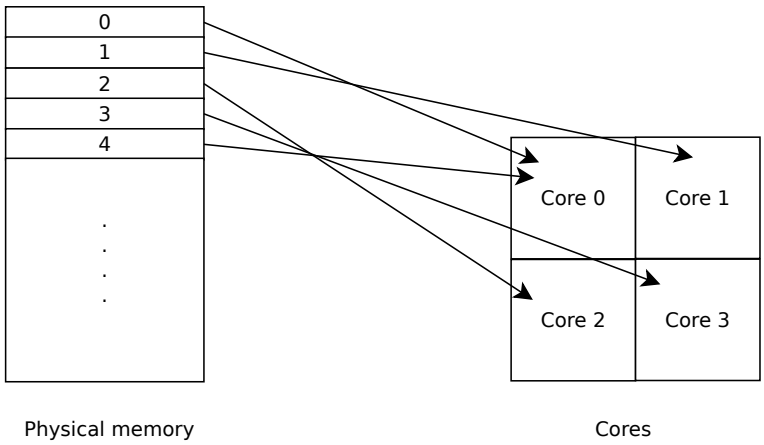


Figure 5.1: Blocks of physical memory are assigned to home nodes present on the cores. The blocks are uniformly distributed in an interleaved manner.

power consumption of the system. This is true regardless of implementation choices, such as the cache-coherence protocol or whether the last-level caches are shared or private.

In the standard hardware implementation, blocks of physical memory are mapped uniformly among all home nodes. The home node H of a given address A is determined by $H = A \texttt{ mod } N$ [21], where N is the number of nodes (see Figure 5.1). The advantages of this method are two-fold. First, the home node of an address can be found using a `mod` computation, which requires simple hardware. Second, because blocks are uniformly distributed among all nodes, it balances the load, thus increasing cache utilization and helping to prevent hotspots.

The above approach is suboptimal, however, because it ignores any underlying structure in the communication pattern of the application. Since blocks are uniformly distributed, a core has on average an equal probability of communicating with any home node, resulting in traffic that is uniform and random. To exploit locality and thus improve the performance and energy efficiency of the system, blocks of

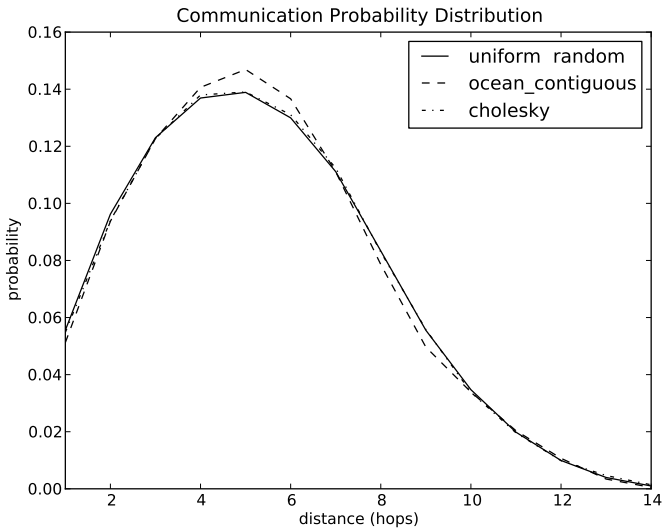


Figure 5.2: Communication probability distribution for two benchmark applications compared to uniform random traffic. Data were collected on a system with standard (uniform) mapping

data must be mapped to home nodes that are located close to where they are most frequently accessed, thereby minimizing the total communication distance. We will refer to the process of optimizing the mapping of memory blocks to home nodes as the *data placement problem*.

To illustrate this point, Figure 5.2 shows the communication pattern of two applications running on a 64-core machine compared to purely uniform-random traffic. The graph displays the Communication Probability Distribution (CPD) [85], that is, the probability that a packet will travel a certain number of hops in a given application. More hops imply less communication locality. [1]

[1]Uniform random traffic does not have a uniform distribution in Figure 5.2 because of the 2D mesh topology. A node may send packets to other nodes with uniform probability, but each node has at most 4 neighbors 1-hop away, 8 neighbors 2-hops away, and so on, until the number of neighbors decreases as the boundaries of the mesh are reached. Therefore, the distribution reflects the average number of neighbors at a certain distance away from the node.

It is important to distinguish between data placement and the thread mapping problem. In the latter, threads, not memory blocks, are mapped to cores in order to optimize some objective function, such as runtime, energy, communication volume, etc. [75]. Here, we assume the location of the threads has already been defined (see Section 5.5.1) and study the impact of different strategies for mapping data to cores. Notice that, if data placement is not optimized, little gain is obtained from thread mapping, since the resulting traffic will be uniform and random.

5.3 The Communication Graph

Communication in a shared-memory system can be modeled as a network of blocks and threads, in which links correspond to messages exchanged between them. Every message to and from a home node is associated with an address that determines the memory block, and a core, which defines the thread. There is no direct communication between threads or between blocks. We define a communication graph $G = \{V, E\}$ as a weighted, undirected bipartite graph in which each vertex corresponds to a block (B) or a thread (T), and edges connect blocks to threads, where the weight $w_{i,j}$ is the total communication (in bytes) between block B_i and thread T_j, summed up over the entire computation. Figure 5.3 depicts a communication graph.

We extracted the communication graph of parallel applications by generating a trace of all messages sent over the network on chip. Each message is represented as an edge, where the size of the message is the weight of the edge. Multiple messages between the same source and destination do not create a new edge, but are used to increase the weight of an existing edge.

An analysis of the network structure of graph G could reveal relevant information about an application. One important metric is the degree of a block, i.e., the number

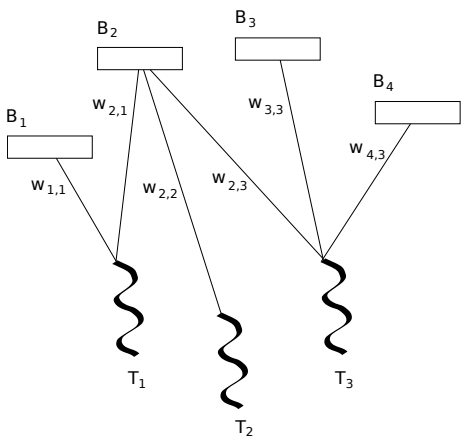


Figure 5.3: Simple illustration of a communication graph. There is no communication between two threads or between two blocks, only between a thread and a block.

of edges connected to the block in the graph, which is related to its level of sharing. A block that has degree N, where N is the total number of threads, is shared by all threads. In this case, not much optimization can be done because the block has no affinity to any specific thread (this is not necessarily true if the weights differ considerably). The best location for such blocks would be in the central nodes of the mesh. On the other hand, if the block degree is 1, then the block is private, i.e., it is only accessed by one thread. This is the best case, because the block can be assigned to the directory at the core in which the thread is running.

Figure 5.4 gives an example of a typical degree distribution of the memory blocks of an application running on a 64-core machine. The figure indicates that the great majority of blocks have very small degree and, therefore, there is potential for optimization in this application. The figure may be misleading in that no blocks seem to exist with large degree. This happens because the number of blocks span several orders of magnitude. Figure 5.5 shows the same distribution in a semi-log plot, which in this case gives a clearer picture of the entire distribution. Notice a peak in the distribution when the degree is 64, corresponding to blocks that are shared by all threads.

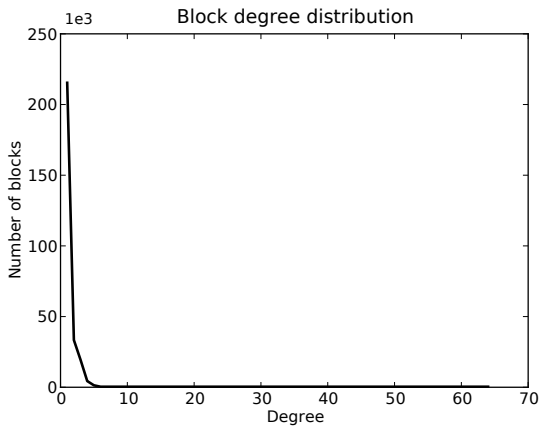


Figure 5.4: Degree distribution of blocks for the `ocean_contiguous` application.

The block degree distribution should not be interpreted as the only property influencing optimization. Several other factors also play an important role. For example, blocks can have different strengths (i.e., the sum of the weights of all links connected to a block) and in general the higher the degree the higher the strength of the block. The positions of the threads on the chip also have an impact on the end result. If a block has small degree but the threads it is connected to are located at a long distance apart from each other, the gains from optimization will be limited.

5.4 An Exact Algorithm for Optimized Data Placement

In this section, we describe our algorithm for optimized data placement. Assuming the communication graph of the application is known, our method finds a placement of blocks to nodes by trading-off locality and load-balancing to minimize energy consumption and improve performance. To motivate the use of load-balancing, we first describe a greedy approach for locality optimization.

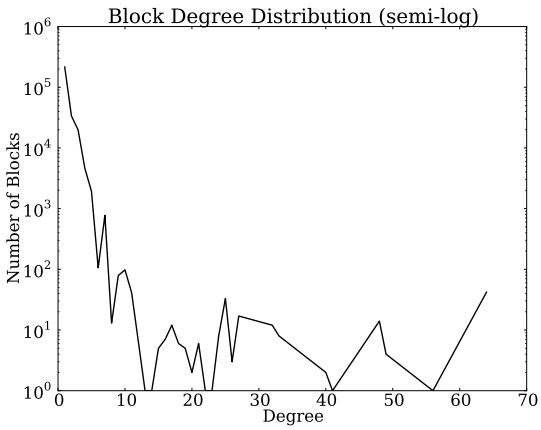


Figure 5.5: Semi-log plot of the block degree distribution for the `ocean_contiguous` application.

5.4.1 Greedy Approach

The simplest approach to locality optimization will greedily assign each memory block to the location on the chip that minimizes the cost of communication. We define the communication cost of a block i assigned to the home node at location p as the weighted sum of the distances between p and each thread j in the communication graph G:

$$C_{ip} = \sum_{j=1}^{N} w_{ij} \cdot d_{pj}, \tag{5.1}$$

where d_{pj} is the distance between position p and the core running the thread j, and N is the total number of cores. The weight w_{ij} corresponds to the total communication between the block and the thread. Its value is zero if there is no communication between them. Using this cost equation, the greedy algorithm works by computing the cost of assigning a block to each of the N locations on the chip, and mapping the block to the location with the minimum cost.

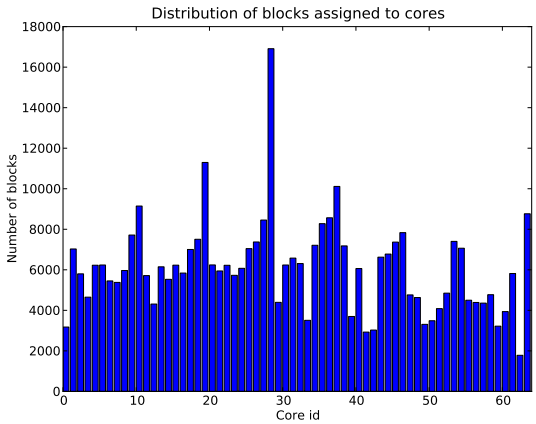


Figure 5.6: Distribution of the number of blocks assigned to each core for the `cholesky` application using a greedy approach.

Although it minimizes communication distances, this greedy approach has a critical flaw: it tends to generate unbalanced mappings where some nodes are assigned many more blocks than others. To illustrate this, we generated a greedy mapping for the `cholesky` application on a 64-core system. Figure 5.6 depicts the number of blocks assigned to each core resulting from the mapping. The figure shows a highly uneven distribution where core 28, for example, is assigned 9.5× more blocks than core 62.

Unbalanced mappings make poor use of the cache capacity of nodes and can severely degrade performance. We ran the `cholesky` application with the greedy mapping of Figure 5.6 and compared the results with the uniform mapping described in Section 5.2, which generates a perfectly balanced load but has no locality optimization. As expected, the resulting communication locality for the greedy method was much higher, with an average communication distance of 2.5 hops, versus 5.3 hops for the uniform method. However, due to unbalanced load and, consequently, an increased number of cache capacity misses, the total network traffic was 290% higher for the greedy method, resulting in runtime and energy consumption that

were 264% and 22% higher, respectively.

This example shows that simply optimizing locality is not sufficient to reduce the energy consumption of the system and may also degrade performance due to underutilization of cache capacity. Below, we describe a formal model for the data placement problem, which optimizes communication locality while balancing the load on each node in order to increase cache capacity utilization.

5.4.2 Description of the Model

Using the same notation as above, we define the communication cost of a block as

$$C_i = \sum_{p=1}^{N} \pi_{ip} \sum_{j=1}^{N} w_{ij} \cdot d_{pj}, \tag{5.2}$$

where

$$\pi_{ip} = \begin{cases} 1 & \text{if block } i \text{ is in position } p, \\ 0 & \text{otherwise.} \end{cases} \tag{5.3}$$

The total communication cost of the system is given by the sum of the costs of all blocks,

$$C_{total} = \sum_{i=1}^{B} C_i, \tag{5.4}$$

where B is the total number of blocks in the application. We now define the following load-balancing constraints:

$$\sum_{p=1}^{N} \pi_{ip} = 1 \tag{5.5}$$

$$\sum_{i=1}^{B} \pi_{ip} \leq K. \tag{5.6}$$

Equation 5.5 simply states that a block can only be assigned to a single position. Equation 5.6 states that the number of blocks assigned to each node must be smaller than or equal to the capacity constraint K, where $K = \lceil B/N \rceil$ for a perfectly balanced load, or K is the cache capacity of a node, if the application fits in cache. Using the equations above, we define an Integer Linear Programming model (ILP) for data placement in multi-core systems as

Optimize:

$$\min_{\Pi} C_{total} : \sum_{i=1}^{B} \sum_{p=1}^{N} \pi_{ip} \sum_{j=1}^{N} w_{ij} \cdot d_{pj}$$

Subject to:

$$\sum_{p=1}^{N} \pi_{ip} = 1$$

$$\sum_{i=1}^{B} \pi_{ip} \leq K$$

$$\pi_{ip} \geq 0 \qquad \forall i \in \{1, 2, \ldots, B\} \text{ and } \forall p \in \{1, 2, \ldots, N\},$$

where the goal is to find the placement matrix Π containing all the variables π. This model is guaranteed to find the maximum communication locality that satisfies the capacity constraints.

Because ILP is **NP**-hard, the above model may not be computationally tractable for large applications with hundreds of thousands, or even millions, of blocks. However, this particular formulation can be solved in polynomial time because its constraint matrix is totally unimodular (the proof of total unimodularity is given in Appendix B). A totally unimodular constraint matrix allows continuous variable values, but there is always an optimal solution in which the variables are integer [82]. Using this result, we define a relaxed version of the problem where the variable π takes continuous values and can be solved in polynomial time with any linear programming technique, such as the simplex algorithm.

5.5 Experimental results

In this section, we present the results of our placement algorithm, referred here as Locality + Load-Balancing (LLB). We first compare the energy and runtime results of our method with first-touch, a commonly used data placement policy for shared-memory systems, briefly described in Section 5.5.1, and show that LLB outperforms first-touch in all analyzed applications. We use the uniform mapping described in Section 5.2 as the base of comparison between the two. We extend our results by performing a sensitivity analysis of LLB, studying the impact of the input data on the topology of the communication graph and the quality of the mappings. We begin this section by describing the simulation methods and system configuration.

5.5.1 Simulation setup

Full-system simulations were performed with the Graphite parallel multi-core simulator [68]. The simulations were performed with in-order, single issue cores. The L1-I and L1-D caches are 4-way set-associative with 32 KB cache-capacity, and 64-byte blocks. The L2-cache is 8-way set-associative with 512 KB capacity, and 64-byte blocks. The directories are full-map with no broadcast and use cache-line granularity. The directory caches are 16-way set-associative with 16384 entries each, and the MESI cache-coherence protocol was used. Although our placement algorithm is applicable to both shared and private caches, our experiments used private caches, which is this the only cache configuration simulated in Graphite.

Energy consumption in the network on-chip was measured with Orion-2 [48], which is included with Graphite. Each hop on the 2D-mesh network takes one cycle, and dimension-order routing was used as the routing algorithm. All simulations were performed on a 64-core system, and runtime and energy were measured after the initialization phase of applications. The number of threads in each application

is the same as the number of cores. As threads are spawned, the simulator assigns each new thread to the next available core, in order. Only one thread is assigned to each core.

The parallel applications used in the simulations are POSIX Threads implementations of the modified SPLASH-2 benchmark [95]. The input of most applications is defined by a random number generator. To produce new inputs, we varied the seed of the generator. An exception is the `ocean` application, which has no input. In this case, we introduced variation by changing the parameters of the application, such as the error tolerance.

The linear programming data placement problem was solved using the `lp_solve` [8] package. Solutions were computed using an Intel Quad-core, 2.83GHz processor and took from 51 seconds for the smallest application (`FFT`), with 30 thousand data blocks, and 21 minutes for the largest one (`cholesky`), with 400 thousand blocks. The first-touch policy works by assigning a block of data to the first node that accesses it during the execution of the application. This policy was implemented to take place only after the initialization phase of applications.[2]

5.5.2 Results

In this section, we present the results of our placement algorithm for 10 scientific benchmark applications. Table 5.1 shows the percent improvement in energy consumption and runtime after running the applications with the LLB and first-touch placements. Improvements are reported relative to the uniform mapping described

[2]A naive policy allocates pages on a first-touch basis from the start of the program execution. This is a problem for applications where one thread initializes everything before processing begins, because all the pages end up on the same node. In our implementation, shared-memory pages are only permanently allocated to nodes once parallel processing has commenced [81].

Table 5.1: Percent improvement in energy and runtime for the LLB and first-touch (FT) data placement (relative to the uniform mapping). Also shown is the total traffic for each method in number of messages.

	Energy Sav. (%)		**Runtime Imp. (%)**		**Total Msgs.**	
Application	LLB	FT	LLB	FT	LLB	FT
`barnes`	45.3	30.0	7.0	6.2	2.1×10^6	2.1×10^6
`cholesky`	40.0	−12.4	2.1	−255.9	5.4×10^6	19.9×10^6
`FFT`	51.8	43.1	5.5	−63.2	3.8×10^5	13.1×10^5
`LU_c`	37.4	−623.9	0.5	−208.2	1.2×10^6	17.7×10^6
`LU_nc`	53.0	28.6	3.8	0.4	2.3×10^7	2.3×10^7
`ocean_c`	74.5	65.9	7.2	2.9	2.0×10^6	2.2×10^6
`ocean_nc`	84.3	83.2	16.9	16.2	1.4×10^7	1.4×10^7
`radix`	57.2	36.0	10.4	−119.1	2.4×10^6	8.7×10^6
`water_ns`	28.0	10.5	0.5	0.4	1.1×10^6	1.1×10^6
`water_sp`	26.7	6.4	0.3	−3.6	3.7×10^5	3.9×10^5

in Section 5.2. Also shown is the total network traffic generated by each method. A high traffic volume is associated with poor load-balancing. Figures 5.7 and 5.8 show the normalized energy and runtime of LLB and first-touch.

Table 5.1 shows large reductions in energy consumption for LLB of up to 84.3% and of 49.8% on average. The obtained runtime improvements were as high as 16.9%, and 5.1% on average. As shown in Table 5.1 and Figures 5.7 and 5.8, in all cases LLB outperformed first-touch, though for some applications, as in `ocean_non_contiguous`, they achieved similar improvements. In some cases, first-touch performed poorly, and worse than the uniform mapping, due to lack of load-balancing. The most extreme case is `LU_contiguous`, in which the total traffic was increased by 14× and the energy consumption by 6×, while the system slowed down by a factor of 2×, compared to the uniform mapping.

Even when the total traffic is approximately the same, the results of first-touch are significantly inferior to those of the LLB mapping, as in the case of `barnes`, `water_nsquared`, and `water_spatial`. This happens because first-touch always al-

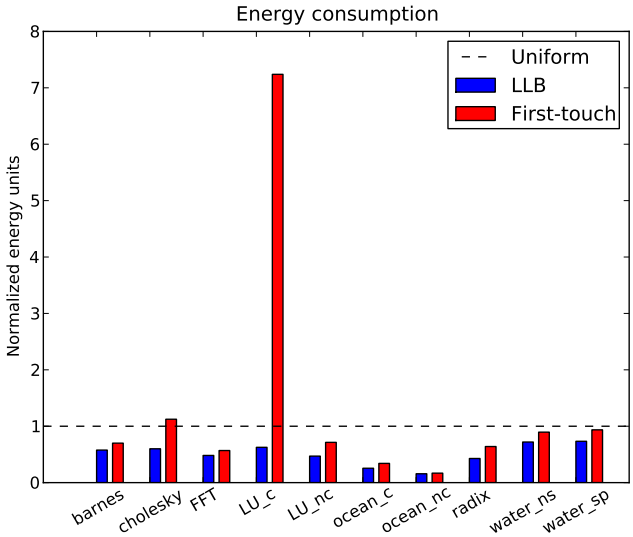


Figure 5.7: Energy consumption of LLB and first-touch normalized by the energy consumption of the uniform mapping.

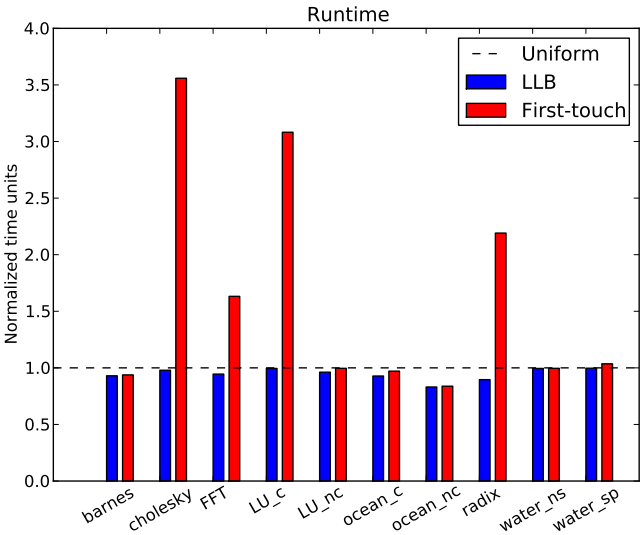


Figure 5.8: Runtime of LLB and first-touch normalized by the runtime of the uniform mapping.

locates a block to one of the sharers, even when this decision is suboptimal. In some cases, the minimal communication distance is achieved by placing a block at a node other than the sharers, but which is physically located in between them. Because LLB is free to choose any node as the location of the block, it achieves smaller communication distances than first-touch.

Figure 5.9 depicts the CPD of the traffic patterns of applications before and after optimization with LLB. As presented in the previous chapter, the CPD corresponds to the probability that messages will travel a certain distance (in number of hops) on the network on chip. The figure shows a shift in the curves towards increased communication locality after the mapping is optimized. The optimized curves vary significantly between applications, suggesting that some of them allow more locality exploitation than others. As will be discussed in Chapter 6, this difference is related to the topology of the different communication graphs.

5.5.3 Sensitivity Analysis

We performed a sensitivity analysis of our method by testing the impact of changing the input data on the results of the LLB mapping. The results in Table 5.2 were produced by extracting the communication graph for a given input, generating the optimized block mapping, and then testing the mapping by running the application on different input data of the same size. Also shown is the similarity between the communication graphs of the different inputs. For two graphs $G_1 = \{V_1, E_1\}$ and $G_2 = \{V_2, E_2\}$, their similarity coefficient was computed as $|E_1 \cap E_2| \ / \ |E_1 \cup E_2|$, which yields a value of 1 if the graphs are identical and 0 if there is no common edge between them. The table shows the average results obtained with 20 randomly generated inputs for each application.

The table shows high similarity of the communication graph between runs for

Table 5.2: Average percentage energy savings and runtime improvement for LLB when testing a previously generated mapping on 20 different inputs. Also shown is the similarity between the communication graphs.

Application	**Energy Sav. (%)**	**Runtime Imp. (%)**	**Graph Similarity**
`barnes`	**45.2**	**4.6**	**0.781**
`cholesky`	40.0	2.1	0.996
`FFT`	51.8	5.6	0.995
`LU_c`	37.2	0.5	0.999
`LU_nc`	53.0	3.7	0.999
`ocean_c`	74.5	7.2	0.997
`ocean_nc`	84.3	16.9	0.998
`radix`	**21.6**	**8.8**	**0.308**
`water_ns`	28.2	0.5	0.982
`water_sp`	26.7	0.3	0.999

most applications. As a result, the energy and runtime savings obtained are approximately the same as those of Table 5.1. Two exceptions are `barnes`, an n-body simulation, and `radix`, a sorting algorithm, for which the similarity coefficients are 0.781 and 0.308, respectively. Interestingly, for `barnes` only a small decay in performance was observed. For `radix`, the improvements were smaller, though the mapping still achieved significant energy and performance savings.

The results of this section show that, for most of the analyzed applications, the mapping only needs to be generated once and, after it is produced, it can be used on multiple runs. Moreover, for applications that perform the same computation over many iterations, it is possible to collect information about the communication graph in the first few iterations and generate the mapping that will be used in the remainder of the application's execution.

5.6 Discussion

Most existing data placement methods for shared-memory multi-processors are heuristics. Our placement method, based on locality maximization and load-balancing, is exact, and performed well for all analyzed applications, outperforming first-touch and the uniform mapping. The proof provided in Appendix B shows that the solution to our method can be found in polynomial time.

The results of Table 5.2 assumed that the two inputs used in the runs were of the same size. Applying a previously generated mapping on an input of different size is complicated because the block addresses may no longer be aligned. However, there are compiler techniques that could be used to make the mapping independent of the size of the variables [13].

The LLB algorithm can be used with different levels of data granularity. In our experiments, we used the cache line granularity, which is the finest possible granularity level. An interesting direction would be to experiment with page granularity, in which case data on the TLB misses can be used to build the communication graph and the operating system can perform the mapping from virtual to physical addresses [21].

5.7 Conclusion

This chapter presented a method for data placement optimization in shared-memory chip multi-processors. The method reduces communication energy consumption by improving locality and cache capacity utilization. The results on scientific benchmarks show a large reduction in NoC energy consumption with significant performance gains. Compared to other approaches in the literature, our method has the advantage of being exact, of focusing on energy consumption, and of providing solu-

tions that are specifically tailored to each application.

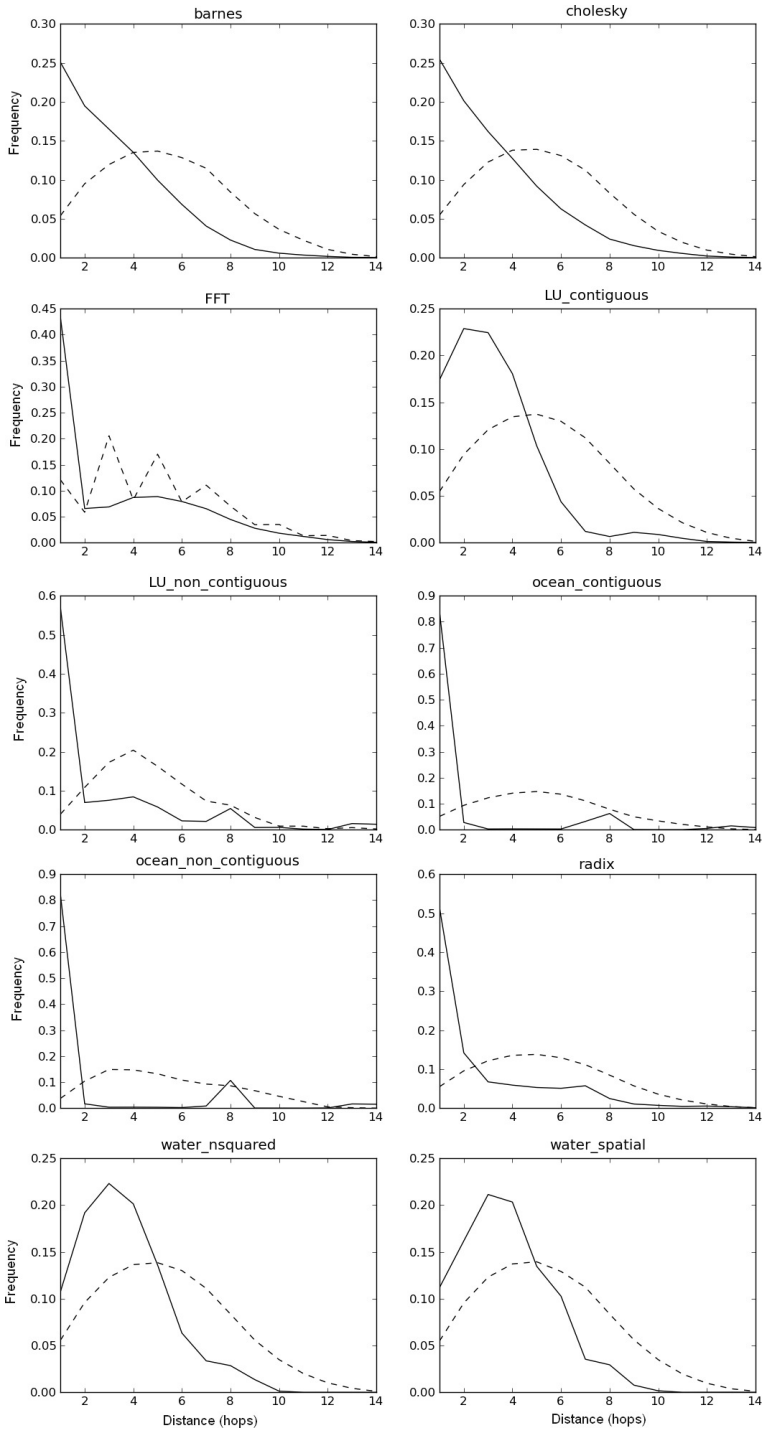


Figure 5.9: Communication probability distribution of benchmark applications before (dashed line) and after (solid line) optimization with LLB.

Chapter 6

Theoretical Analysis of NoC Energy Consumption

In the previous chapter, we saw that some applications have more potential for energy optimization than others. For example, `ocean_non_contiguous` had 84.3% energy reduction, while `water_spatial` had only 26.7%. In this chapter, we use Rent's rule to show that the minimum energy consumption of applications is constrained by the structure of their communication graph. The higher the Rent's exponent—or fractal dimension—of a graph, the lower its communication locality and the higher its energy consumption.

The theoretical analysis of energy consumption has multiple applications to hardware and software design. It can be used to evaluate the potential for communication locality in an application prior to execution and, therefore, assess the quality of the data placement algorithm being used; aid in the design of more energy-efficient applications; and provide first-order predictions of energy consumption for new applications and new systems (e.g., a multi-core chips with a larger number of cores).

In Section 6.1, we apply the bandwidth version of Rent's rule to the commu-

nication graph of parallel applications and measure the Rent's exponent (p) and coefficient (b). The Rent's rule parameters are then used to estimate the communication locality of applications in Section 6.2, and to predict the minimum NoC energy consumption in Section 6.3. The theoretical predictions are compared with simulation results.

6.1 Rent's Rule for Multi-Core Systems

As presented in Chapter 4, the bandwidth version of Rent's rule is given by

$$B = bN^p, \tag{6.1}$$

where B is the bandwidth sent or received by a cluster of N network nodes, b is the average bandwidth per node, and $0 \leq p \leq 1$ is the Rent's exponent. A communication graph follows Rent's rule if its behavior in a log-log plot of N vs. B can be approximated by a straight line, where the slope of this line is the Rent's exponent. The Rent's exponent is monotonically related to the fractal dimension D of a communication graph as $p = (D-1)/D$ [86]. Therefore a higher Rent's exponent implies higher fractal dimension.

To measure p and b, we partitioned the communication graph of the benchmark applications into two clusters of equal size, and measured total weight of the cuts (or bandwidth). For each resulting subgraph, the process was repeated, until there was a single node per cluster. Figure 6.1 shows the curve obtained for the size of a cluster versus the average bandwidth for three applications. Similar behavior was obtained for the other applications (curves not shown). The saturation of the curves for large module sizes is known as Region II of Rent's rule [86].

Rent's rule approximations shown in the figure were obtained using a linear fit with a cutoff at cluster size 8. As cluster sizes increase beyond this point, the Rent's

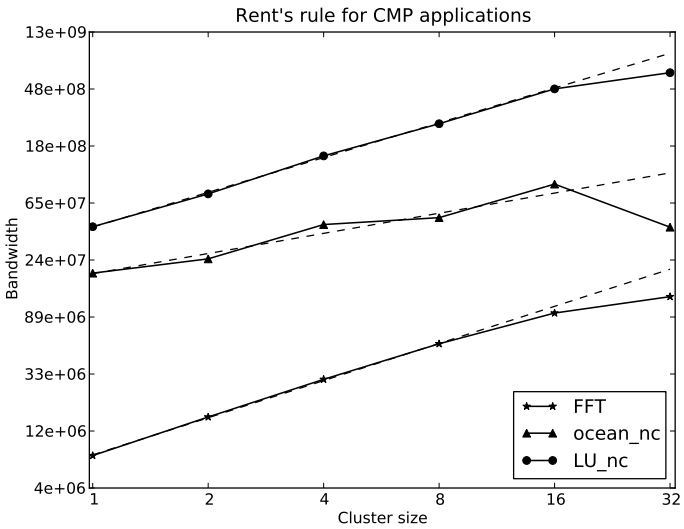


Figure 6.1: Rent's rule for three CMP applications.

rule behavior begins to saturate. This saturation is known as region II of Rent's rule [86] and occurs as the module sizes approach the size of the entire system. For systems of larger size, the portion of the curve corresponding to region II tends to decrease relative to region I.

Using a linear approximation, we extracted the Rent's exponent p and coefficient b for all 10 applications, as shown in Table 6.1, where a wide variation in the Rent's exponent from 0.36 up to 0.94 is observed.

6.2 Modeling Communication Locality

In this section, we use the Rent's exponent to estimate an upper-bound on communication locality. Based on the Wire Length Distribution (WLD) model of [25], which was initially developed for VLSI circuits, we compute the average distance traveled by a message in different applications. Equation 6.2 defines the probability of having

Table 6.1: Rent's rule parameters p and b for CMP applications.

Application	p	b (Bytes)
`barnes`	0.87	4185561
`cholesky`	0.89	10112351
`FFT`	0.94	791009
`LU_contiguous`	0.89	2695664
`LU_non_contiguous`	0.81	45880087
`ocean_contiguous`	0.36	13627773
`ocean_non_contiguous`	0.51	19008764
`radix`	0.65	6207811
`water_nsquared`	0.92	746057
`water_spatial`	0.91	2238112

a wire connecting two logic gates with Manhattan distance d. We use this equation to represent the probability of communication between cores, where N is the number of cores on a square mesh network.

Region I: $1 \leq d < \sqrt{N}$

$$P(d) = \frac{\Gamma}{2N\left(1 - N^{p-1}\right)} \left(\frac{d^3}{3} - 2\sqrt{N}d^2 + 2\sqrt{N}d \right) d^{2p-4}$$

Region 2: $\sqrt{N} \leq d < 2\sqrt{N} - 2$

$$P(d) = \frac{\Gamma}{6N\left(1 - N^{p-1}\right)} \left(2\sqrt{N} - d \right)^3 d^{2p-4} \tag{6.2}$$

where Γ is a normalization constant. From the above formula, the average communication distance is computed as the weighted sum of the probabilities with their respective distances as

$$\overline{d} = c \cdot \sum_{d=1}^{2\sqrt{N}-2} d \cdot P(d), \tag{6.3}$$

where c is a constant to be determined. Using linear regression, we found the constant c that maximizes the fit of the model to the data. Figure 6.2 shows the results of the model using the Rent's exponents given in Table 6.1, with $c = 1.38$. The figure

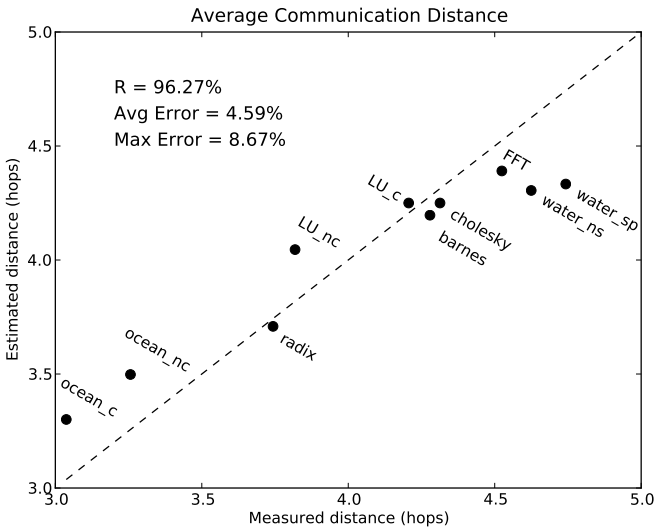


Figure 6.2: Measured and estimated communication distance for CMP applications. The dashed line indicates perfect agreement between empirical and theoretical values.

compares predicted and measured average communication distance of applications after optimization by the placement algorithm described in Section 5.4.

The results of the model agree closely with the empirical values, with an average error of 4.58% and maximum error of 8.64%. The correlation coefficient of 96% shows that, even at this relatively small scale, the Rent's exponent can explain most of the variation in communication distance and is, therefore, a good predictor of communication locality in CMP applications. The agreement between theory and experiment also validates the ability our method to exploit locality.

The intuition behind these results is that the lower the dimensionality, the more independent the modules of the graph are from each other, allowing more freedom for blocks to be placed close to where they are most frequently used. For high-dimensional graphs, there is more interdependence between modules and less opportunity for optimization. A worst-case scenario is a graph with Rent's exponent 1, in which all nodes are equally connected to each other. In this case, no matter how

this graph is placed, there is never any improvement in communication locality.

6.3 Modeling Energy Consumption

The energy consumption of the interconnect is easily computed from the estimated average communication distances. The energy used by a message of length l (in bytes) when traversing one hop on a 2D-mesh NoC is given by

$$E_{hop}(l) = E_{router}(l) + E_{link}(l), \tag{6.4}$$

where E_{router} and E_{link} are the average energy used by the message when traversing a router and a link, respectively; their values can be obtained from power simulators, such as Orion 2. Because the network carries messages of different sizes, the average energy used per byte can be obtained by

$$E_{hop}(1) = \frac{\sum_l E_{hop}(l) \cdot N_l}{\sum_l l \cdot N_l}, \tag{6.5}$$

where N_l is the number of messages of size l. The above conversion is necessary because arbitration occurs only for the header of a message and, therefore, energy consumption is not directly proportional to message size. Using the average distance traveled by a message (Equation 6.3), the energy used by a byte when traversing a hop (Equation 6.5), and the average number of bytes per node (the parameter b of Rent's rule, given in Table 6.1) the total energy of the application can be calculated:

$$E_{total} = \overline{d} \times E_{hop}(1) \times N \times b, \tag{6.6}$$

where $N \times b$ is the total number of bytes sent and received over the network.

The energy predicted by the model was compared with the measured energy consumption of applications, as shown in Figure 6.3. The results have a high correlation coefficient of 99.82%, with average error of 4.53% and maximum error of 8.64%.

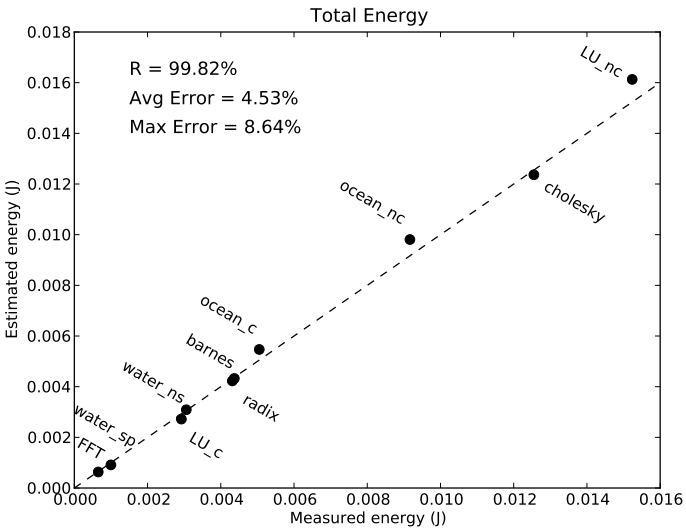


Figure 6.3: Measured and estimated energy consumption of CMP applications. The dashed line indicates perfect agreement between empirical and theoretical values.

6.4 Related work

Very few related models exist for predicting energy consumption in parallel architectures. A model for the impact of Dynamic Voltage Scaling (DVS) on energy consumption is proposed in [79]. Using linear programming, the authors establish an empirical lower bound on the energy consumption of applications with DVS. In reference [59], a model for energy consumption based on the computation to communication ratio of applications is presented. This model is used to predict the optimal number of cores for toy parallel applications, but no experimental verification is provided. A model for the energy-time trade-offs in generic computer architectures is proposed in [9]. In this highly abstract representation, they use a task graph to define theoretical lower-bounds on the time and energy of sorting, binary addition, and multiplication problems. Although an interesting theoretical development, the authors conclude that the abstraction level is not yet appropriate for developing practical algorithms.

Our work is the first to predict the energy consumption and communication locality of parallel applications from the topology of the communication graph. Our approach provides an elegant explanation to why some applications have more potential for communication locality than others. Because we use Rent's rule, only a compact description of the graph, based on the Rent's exponent p and coefficient b, is required in the modeling.

6.5 Conclusion

This chapter presented a theoretical analysis of energy consumption and communication locality of parallel applications based on Rent's rule. Our analysis reveals why some applications have more communication locality than others. Although it may seem obvious that locality is constrained by the topology of the communication graph, defining the relevant topological parameters and using them to estimate locality is non-trivial. The close agreement between theoretical and empirical values helps validate the LLB algorithm, and verifies its ability to exploit communication locality and reduce energy consumption. Finally, the analysis presented here could be used to provide first-order predictions of energy consumption of new applications and/or new systems. This is especially relevant when we consider the energy that will be used in future multi-core chips with hundreds of cores. Our framework is not limited to CMPs and could be extended to arbitrarily large machines composed of multiple chips.

Chapter 7

A General Power-Performance Scaling Law for Computing

Digital computers scaled in size over 6 orders of magnitude in the last 40 years. If growth continues as expected, in 10 years chips will have trillions of transistors, more than the number of neurons in the human brain [3]. But what will be the power consumption of computers in the future? How many cores will they have? What will be their performance and clock frequency? Currently, we have no answers to these questions.

In computer science, there is no general theory that explains the scaling of computing hardware. Although certain well-known patterns exist which serve as guidelines for technology roadmaps (such as Moore's law and Koomey's law [58]), these empirical observations lack theoretical explanation. Amdahl's law, a theoretical bound for the speedup of parallel algorithms, has no counterpart in terms of power consumption, and does not model communication, the dominant factor in the scaling of power and performance in modern architectures [42]. Currently, computer architecture design is mostly and empirical practice, with a few guiding principles and no

general theory of scaling.

In biology, a general theory exists that explains the scaling of power consumption (metabolic rate) as a function of body mass in organisms. From unicellular microbes to giant whales and trees, biological organisms scale over 21 orders of magnitude — vastly more than computer architectures. Despite the amazing diversity and complexity of organisms, their metabolism manifests an extraordinary simplicity when viewed as a function of size. From a few basic principles, Metabolic Scaling Theory (MST) in biology explains this pattern as a result of geometric constraints imposed by the fractal branching structure of vascular networks. This theory predicts, with remarkable accuracy, how metabolic rate scales across species and during the lifetime of a single organism.

By combining the results from MST in biology and Rent's rule in VLSI design, this chapter develops a theory for the scaling of power and performance in computer architectures. We view the scaling of computers as a geometrical process, in which spatial constraints determine how fast communication, power, and throughput increase as a function of size. Assuming that computer architectures are optimized to minimize the energy-delay product [33, 1], we determine the optimal scaling dimensions of an idealized computer logic network that has minimum cost. The geometry of this network leads to extremely simple power-performance scaling laws that accurately describe the scaling of power and performance in microprocessors over a range of several orders of magnitude.

This chapter is organized as follows. In Section 7.1, we analyze the scaling of vascular systems and digital circuits by comparing MST and Rent's rule, and identify three dimensions in which networks scale. In Section 7.2, we propose a unified model of network scaling that incorporates properties of both MST and Rent's rule. This model is then used in Section 7.3 to derive general allometric scaling relations for networks, such as network volume and total wire length. In Section 7.4, these

allometric relations are used to derive general expressions for the scaling of resistance, capacitance, latency, and bandwidth, and from these expressions determine the conditions for optimal energy-delay product. Finally, in Section 7.5, we present general power-performance scaling laws for computing and compare their predictions with real-world data. A discussion of the implications of our results are presented in Section 7.6, and Section 7.7 concludes the chapter.

7.1 The Scaling of Vascular Systems and Digital Circuits

7.1.1 The West-Brown-Enquist model

Manifesting an extraordinary diversity of form and function over an enormous range from the largest animals and plants to the smallest microbes, life on Earth is the most complex physical phenomenon known to us. Yet, many of the most fundamental biological processes in organisms display striking regularity over an immense range of 21 orders of magnitude [91]. Such regularity is characterized by quarter-power scaling as a function of size. A canonical example is the 3/4 power scaling of metabolism as a function of body mass, known as Kleiber's law (Figure 7.1). This scaling relationship can be written as

$$B \propto M^{3/4}, \tag{7.1}$$

where B and M are metabolism and mass, respectively. Similarly, gestation period and lifespan scale as the $1/4$ power, growth rate as the $-1/4$ power, and heart rate as the $-1/4$ power, among others [14].

In their seminal paper, West, Brown, and Enquist [92] postulate that a common mechanism underlies these laws: Living things are sustained by the transport of ma-

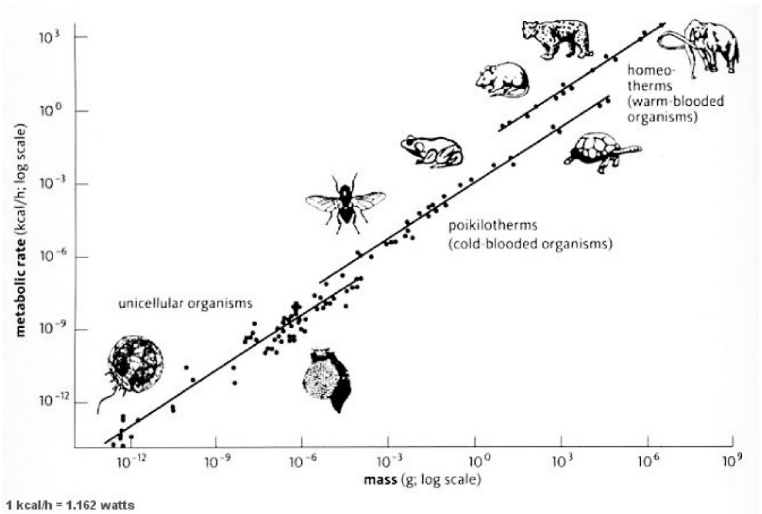


Figure 7.1: Kleiber's law.

terials through hierarchical branching networks that supply all parts of the organism. They present a quantitative model of network growth, known as the *West-Brown-Enquist* (WBE) model, that explains the origin and ubiquity of the quarter-power scaling and predicts essential features of transport systems in biology. The WBE model relies on three basic assumptions or conditions:

1. In order for the internal network to supply the entire volume of the organism, a *space-filling* fractal-like branching pattern is required.

2. The terminal branches of the network, such as the capillaries in the circulatory system, are *size-invariant*, i.e., their size remains constant as the system scales.

3. The energy required to distribute resources is minimized, which results in a network with *area-preserving* branching.

The schematic of the model is shown in Figure 7.2 for a branching factor of two. Each branch of the network has radius (r_i) and length (l_i) that are characteristic of its hierarchical level i. The model is defined by two parameters, i.e., the rate

at which branches become thinner (β) and the rate at which they become shorter (γ) as they move down in hierarchy from the root of the tree towards the leaves. The cross-sectional area-preserving branching condition requires $\beta = b^{-1/2}$, and the space-filling condition requires $\gamma = b^{-1/3}$, where b is the branching factor. More formally:

$$\beta = \frac{r_i}{r_{i+1}} = b^{-1/2} \tag{7.2}$$

and

$$\gamma = \frac{l_i}{l_{i+1}} = b^{-1/3}. \tag{7.3}$$

This model can be used to explain Kleiber's law in the following manner. Assuming that metabolism (B) is proportional to the number of capillaries (leaves of the tree) (N) and that the mass of the organism (M) is proportional to the volume of the network (V), the allometric scaling relationship between metabolism and body mass can be written as:

$$N \propto V^a, \tag{7.4}$$

where a is the scaling exponent. By computing N and V using the parameters of the model, the exponent $a = 3/4$ is easily derived.

Metabolic Scaling Theory (MST) has become an important subfield in biology, and it has significantly influenced other disciplines, such as Complex Systems. Several extensions to the WBE model have been proposed that attempt to incorporate more realistic assumptions and match observed data more accurately [5, 77, 6, 29]. A comprehensive review of MST with all its extensions and applications is outside the scope of this chapter. For the purposes of the work presented here, the most important lesson from the theory is clear: The 3/4 power scaling of metabolism in biological organisms is essentially a geometrical phenomenon.

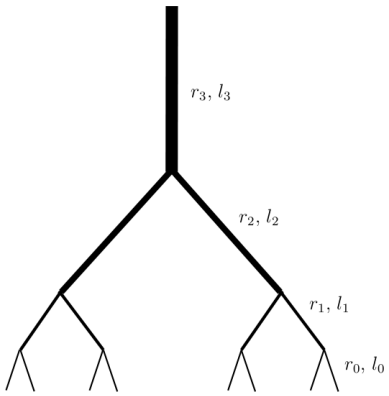


Figure 7.2: Illustration of a fractal branching network structure with branching factor 2.

7.1.2 Rent's rule

In vascular systems, the network branches get thicker and longer as they go up in hierarchy, and this scaling is described by the WBE model. Interestingly, in spite of their differences in function and topology, digital circuits scale in a similar way. Wires in a microprocessor chip are arranged hierarchically in metal layers, and the higher the layer the thicker and longer the wires (see Figure 7.3). As the number of transistors increases, so does the number of metal layers, or hierarchical levels. If wires scale geometrically with the number of nodes in the network, than the scaling of wire thickness and length in chips is also given by the WBE model, although the parameter values are likely different from those for vascular networks.

However, digital circuits scale in a third way that has no analog in vascular systems. This scaling pattern, which has been widely discussed in previous chapters, is Rent's rule. Rent's rule describes how communication between different parts (or modules) of the circuit scales with size. Similar to the scaling of thickness and length of wires, the scaling of communication is also a hierarchical relationship and can be

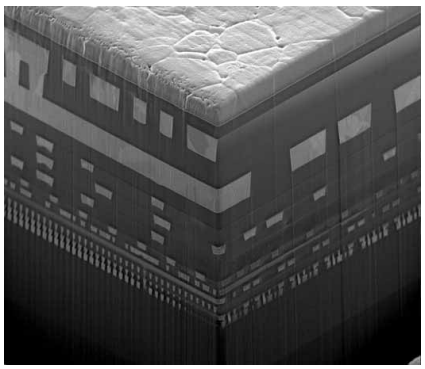


Figure 7.3: Cross section of twelve layers of interconnect. Figure reproduced from [94].

expressed as such. Recall that Rent's rule is given by

$$C(n) = kn^p, \tag{7.5}$$

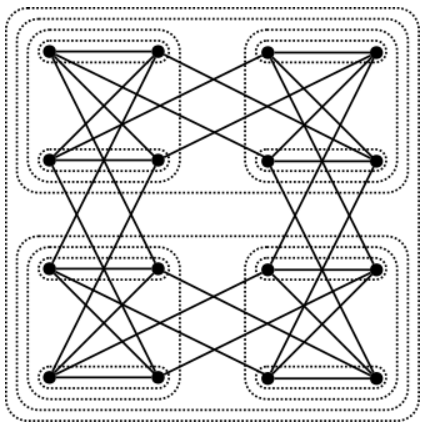
where $C(n)$ is the external communication, n is the size of a module, k is the average external communication of a module with size 1, and p is the Rent's exponent. For a two-way hierarchical partitioning, the size of a module is given as $n = 2^i$, where i is the hierarchical level. Therefore, we can rewrite Rent's rule as

$$c_i = c_0 \cdot 2^{ip}, \tag{7.6}$$

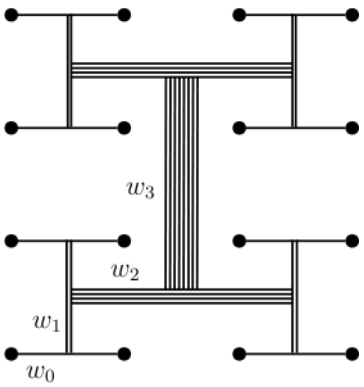
where c_i is the external communication of a module at hierarchical level i and $c_0 = k$.

Like the WBE model, this scaling pattern also defines a fractal branching structure that scales hierarchically with the number of nodes. To show this, we conveniently represent Rent's rule as a fat-tree [63], where leaves of the tree are the nodes of the network, branches are modules, and the width w_i of a branch at hierarchical level i corresponds to the average communication per module at that level. This is illustrated in Figure 7.4, where Figure 7.4a depicts a two-way hierarchical modular decomposition of a network with 16 nodes, and Figure 7.4b shows its corresponding fat-tree representation. As the hierarchical level increases, the width of the branches

in the tree also increase, proportionally to the amount of communication (or number of wires) at each level.



(a) Two-way hierarchical modular decomposition of a network with 16 nodes.



(b) Fat-tree representation of Rent's rule for the same network.

Figure 7.4: Visualization of the hierarchical interpretation of Rent's rule.

Notice that the width of a branch, w_i, is not the same as the external communication, c_i. While c_i correponds to the total external communication of a module, w_i is the communication only at level i (e.g., in the figure, $c_0 = 4$ and $w_0 = 1$). However, as the number of nodes increases, the difference between them vanishes so that both quantities scale in the same way. This is shown by the following theorem:

Theorem 1: For $0 \leq p < 1$, the width of a branch in the fat-tree (w_i), scales proportionally to the external communication of a module (c_i).

Proof: w_i is the average number of wires per module at hierarchical level i. This corresponds to the external communication of a module at level i minus the external communication at level $i+1$ divided by the branching factor. For a generic branching factor b, the relationship between w_i and c_i can be written as [86]

$$w_i = c_i - \frac{c_{i+1}}{b}. \tag{7.7}$$

Solving this equation gives

$$\begin{aligned} w_i &= c_0 \cdot b^{ip} - \frac{c_0 \cdot b^{(i+1)p}}{b} \\ w_i &= c_0 \cdot b^{ip} - c_0 \cdot b^{(i+1)p-1} \\ w_i &= c_0 \cdot b^{ip} - c_0 \cdot b^{ip+p-1} \\ w_i &= c_0 \cdot b^{ip} \left(1 - b^{p-1}\right). \end{aligned} \tag{7.8}$$

For $p < 1$, the expression in parenthesis always evaluates to a positive real. As a result,

$$w_i \propto b^{ip}, \tag{7.9}$$

or

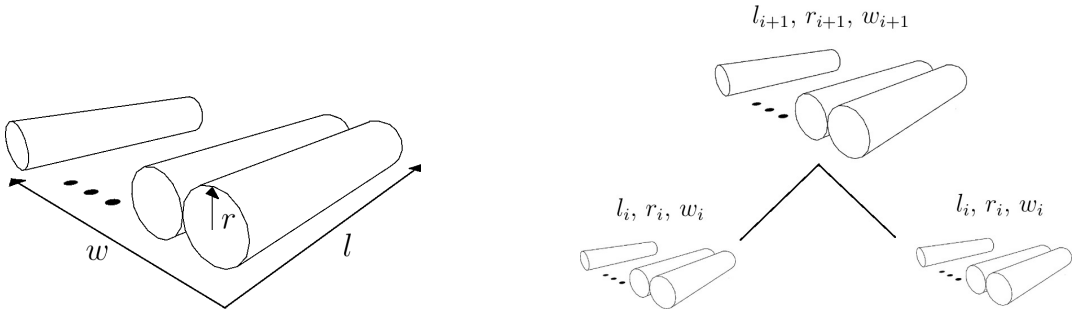
$$w_i = w_0 \cdot b^{ip}. \tag{7.10}$$

Theorem 1 shows that the external communication of a module and the width of a branch in the fat-tree scale with the hierarchical level in the same way. Therefore, the fat-tree structure is a sound representation of Rentian scaling. This result places Rent's rule in the same framework as the WBE model, as the scaling of communication can be modeled as tree structure whose branch dimensions scale geometrically with the number of nodes.

7.2 A unified model of network scaling

In the previous discussion, we saw that interconnection networks scale in three different ways: wire length, wire thickness, and width, and that the WBE model accounts for the first two, while Rent's rule models the third. In this section, we combine the properties of these two models into a single hierarchical model of network scaling.

The unified model consists of a hierarchical branching structure, where each branch is composed of a collection of wires and for which the geometry of a branch is defined by the parameters l (length), r (thickness), and w (width), as shown in Figure 7.5a. As the system scales and the network branches, wires at the lower hierarchical levels get shorter and thinner, and the number of wires per branch decreases. Figure 7.5b illustrates the branching pattern of the model for a branching factor of 2.



(a) Geometry of wires at a given hierarchical level, defined by three dimensions.

(b) Branching structure of the model for a branching factor of 2.

Figure 7.5: Schematic of the hierarchical model of network scaling.

In the model, l, r, and w scale according to well-defined geometric dimensions, Δ_l, Δ_r, and Δ_w, which can have non-integer values. Although real microprocessor systems are not perfectly regular structures, there are supporting evidences that their interconnection networks have a well-defined geometry. In the VLSI literature, the geometric scaling of wire lengths is an approximation commonly used in methods for total wire length estimation [86, 30, 31]. The existence of a well-defined thickness dimension is predicted by Dennard scaling [27], and the geometric scaling of width (or communication) is empirically supported by Rent's rule [61].

We now formally define the proposed hierarchical model by analyzing the scaling of length, thickness, and width. The scaling dimensions introduced in this section are summarized in Table 7.1, together with another dimension that will be introduced

Table 7.1: List of all the scaling dimensions defined in this chapter.

Parameter	Description
Δ_l	Length dimension, or spatial dimension
Δ_r	Thickness dimension
Δ_w	Width dimension
Δ_g	Fractal dimension of the network graph

in later sections[1].

7.2.1 Length

Geometrically, the scaling of edge lengths of a tree determines the fractal dimension of the space that is occupied by its leaves [65]. Because the capillaries of vascular networks must supply blood to all parts of the organism, the lengths of its branches must scale so that the network is *volume-filling*. In the case of computer chips, because the nodes of the network occupy the two-dimensional space, the lengths of wires must scale so that the network is *area-filling*.

For a well-defined geometry, the hierarchical scaling of lengths is written as

$$l_{i+1} = l_i \cdot b^{1/\Delta_l}, \tag{7.11}$$

where l_i is the length of a wire at hierarchical level i, b is the branching factor, and Δ_l is the length dimension. We solve this recursive relation and obtain a closed-form expression for the scaling of lengths as a function of the hierarchical level as

$$l_i = l_0 \cdot b^{i/\Delta_l}, \tag{7.12}$$

where l_0 is the length of the smallest wire. For an area-filling digital circuit network, $\Delta_l = 2$, which is consistent with the modeling of total wire length in the VLSI

[1]We note there is yet a fourth way in which networks scale, that is, the degree distribution of nodes. In Appendix C, we show that this fourth scaling dimension is accounted for by our model as a growth process that is independent of the network geometry. The reading of this appendix is not required for understanding the rest of this chapter.

literature [30]. For a volume-filling vascular network, $\Delta_l = 3$, which is in accordance with the WBE model.

7.2.2 Thickness

The variable r corresponds to a linear measure of thickness, such that wire cross-sectional area scales as $A \propto r^2$. This is independent of the shape of the wire cross section, so the model makes no distinction between, for example, cylindrical or rectangular shaped wires.

For a well-defined geometry, the hierarchical scaling of thickness can be written as

$$r_{i+1} = r_i \cdot b^{1/\Delta_r}, \tag{7.13}$$

where r_i is the thickness of a wire at hierarchical level i, b is the branching factor, and Δ_r is the thickness dimension. We solve this recursive relation and obtain a closed-form expression for the scaling of thickness as a function of the hierarchical level as

$$r_i = r_0 \cdot b^{i/\Delta_r}. \tag{7.14}$$

From the above equation, an area-preserving vascular network has $\Delta_r = 2$. For digital circuits, it is predicted by Dennard scaling that Δ_r is also equal to 2, which corresponds to the ideal case in which both length and thickness scale at the same rate and latency is independent of distance [27]. The optimal value for Δ_r will be discussed in Section 7.4.

7.2.3 Width

The variable w represents the number of wires in a branch of the tree structure, which is given by Rent's rule. Intuitively, a branch corresponds to a module in Rent's rule, and the number of wires per module is the width of a branch (see Figure 7.4b). From Equation 7.10, the scaling of branch widths is given as

$$w_i = w_0 \cdot b^{ip}. \tag{7.15}$$

This defines the recursive relation

$$w_{i+1} = w_i \cdot b^{1/\Delta_w}, \tag{7.16}$$

where Δ_w is the width dimension, and $\Delta_w = 1/p$. The closed-form solution is given as

$$w_i = w_0 \cdot b^{i/\Delta_w}. \tag{7.17}$$

Notice that, for vascular networks, the width of a branch is constant with the hierarchical level (i.e., $w_i \propto 1$). From the above equation, this occurs when $\Delta_w \to \infty$, in which case the width dimension is undefined. For computer chips, $p \approx 0.5$ [4] and, therefore, Δ_w is close to 2. The optimal value for Δ_w will be discussed in Section 7.4.

Hence, we model the geometry of digital circuit networks using the following independent scaling relations:

$$\begin{cases} l_i = l_0 \cdot b^{i/\Delta_l} \\ r_i = r_0 \cdot b^{i/\Delta_r} \\ w_i = w_0 \cdot b^{i/\Delta_w}. \end{cases} \tag{7.18}$$

7.3 Allometric scaling

In biology, allometry is the study of the differential growth of parts of an organism in relation to its size [34]. When the whole organism and its parts scale at the same rate, growth is *isometric*. However, when certain parts scale at a rate that is different from that of the organism, the growth of these parts is *allometric*. In the case of networks, the phenomenon of allometry occurs, for example, when the total volume of the network and the number of nodes grow at different rates.

Here, we use the hierarchical model of network scaling proposed in the previous section to derive general allometric relations of networks and analyze the conditions that lead to allometry. These calculations will serve as the basis for computing the electrical properties of the network, such as resistance and capacitance, in the next section.

7.3.1 Volume

The allometric relation between volume of a network and the number of nodes can be written as

$$V \propto N^{\alpha}, \tag{7.19}$$

where V is the volume, N is the number of nodes, and α is the allometric exponent. When $\alpha > 1$, the size of the network scales faster than the number of nodes, resulting in allometric growth. In order to determine the conditions that lead to allometry, we need to compute N and V, where

$$\alpha = \frac{\log V}{\log N}. \tag{7.20}$$

From the hierarchical model in Section 7.2, the number of nodes is given as

$$N = b^{H-1}, \tag{7.21}$$

where, H is the number of hierarchical levels (or height). The total volume of the network is given by summing the volume of all the branches at each hierarchical level:

$$\begin{aligned} V &\propto l_0 r_0^2 w_0 b^{H-1} + l_1 r_1^2 w_1 b^{H-2} + \cdots + l_{H-1} r_{H-1}^2 w_{H-1} b^0 \\ &\propto l_0 r_0^2 w_0 b^H \sum_{i=0}^{H-1} b^{i\left(\frac{1}{\Delta_l}+\frac{2}{\Delta_r}+\frac{1}{\Delta_w}-1\right)}. \end{aligned} \tag{7.22}$$

From Equation 7.22, there are three possible cases for the scaling of V as a function of H:

Case 1: $\frac{1}{\Delta_l}+\frac{2}{\Delta_r}+\frac{1}{\Delta_w}-1<0$**.** In this case, the summation series converges as H increases, thus $V \propto b^H$, or $V \propto N$. Hence, $\alpha = 1$, and scaling is isometric.

Case 2: $\frac{1}{\Delta_l}+\frac{2}{\Delta_r}+\frac{1}{\Delta_w}-1=0$**.** Here, the summation series diverges, resulting in $H(H-1)/2$. Thus, $V \propto H^2 b^H$, or, equivalently, $V \propto N \log_b^2 N$. In this special case, there is a discontinuity and growth is neither isometric nor allometric. We name this growth *pseudo-allometric*.

Case 3: $\frac{1}{\Delta_l}+\frac{2}{\Delta_r}+\frac{1}{\Delta_w}-1>0$**.** In this case, the last term of the summation series dominates, and $V \propto b^{H\left(\frac{1}{\Delta_l}+\frac{2}{\Delta_r}+\frac{1}{\Delta_w}\right)}$. As a result,

$$\alpha = \frac{1}{\Delta_l}+\frac{2}{\Delta_r}+\frac{1}{\Delta_w}, \tag{7.23}$$

and scaling is allometric.

A special case of allometric scaling occurs when $\Delta_l = 3$ and $\Delta_r = 2$, and $\Delta_w = \infty$, which leads to $\alpha = 4/3$. This is case of vascular systems, in which the network is volume-filling, area-preserving, and there is no Rent's rule scaling. Since $N \propto V^{1/a} = V^{3/4}$, metabolism ($N$) scales as body size ($V$) as the 3/4 power.

7.3.2 Wire length

For electronic circuits, the total wire length is an important quantity. The longer, and the more wires there are, the higher the cost of the network in terms of energy consumption, materials and space. The total wire length is given by summing the lengths of all wires:

$$\begin{aligned} L &= l_0 w_0 b^{H-1} + l_1 w_1 b^{H-2} + \cdots + l_{H-1} w_{H-1} b^0 \\ &= l_0 w_0 b^H \sum_{i=0}^{H-1} b^{i\left(\frac{1}{\Delta_l}+\frac{1}{\Delta_w}-1\right)}. \end{aligned} \tag{7.24}$$

As a result, in the case of allometric scaling,

$$L \propto N^{\frac{1}{\Delta_l}+\frac{1}{\Delta_w}}. \tag{7.25}$$

In order to compute the average wire length, first we prove that the average degree of a node is constant with scaling. The average degree is given by the total number of wires divided by the number of nodes:

$$\begin{aligned} \overline{k} &= \frac{1}{N}\left(w_0 b^{H-1} + w_1 b^{H-2} + \cdots + w_{H-1} b^0\right) \\ &= \frac{w_0 b^H}{N} \sum_{i=0}^{H-1} b^{i(\frac{1}{\Delta_w}-1)}. \end{aligned} \tag{7.26}$$

Since $p \leq 1$, then $\Delta_w \geq 1$, and the above series converges. As a result:

$$\overline{k} \propto \frac{b^H}{N} \propto 1. \tag{7.27}$$

Therefore, the average degree is constant, and the total number of edges is proportional to N. The average wire length is then given as

$$\overline{L} \propto \frac{L}{N} = N^{\frac{1}{\Delta_l}+\frac{1}{\Delta_w}-1}, \tag{7.28}$$

or

$$\overline{L} \propto N^{\frac{1}{\Delta_l}+p-1}. \tag{7.29}$$

The above result is a generalization of the special case when $D = 2$, that is, the wire length for 2D chips. In this case, we obtain

$$\overline{L}_{2D} \quad \propto \quad N^{p-\frac{1}{2}}, \tag{7.30}$$

which matches the result derived by Donath (1979) for the average wire length in VLSI circuits [30].

7.3.3 Fractal dimension

We now show that Rent's rule corresponds to a more general property of networks, i.e., a fractal interconnection topology. This gives rise to an important relation between the fractal dimension of the network graph, Δ_g, and the width dimension, Δ_w, which will be used in the next section to interpret the conditions for which energy-delay is minimized. We propose the following intuitive definition of network fractal dimension:

Definition 1: The fractal dimension of a network graph (Δ_g) is the lowest spatial dimension in which the nodes of the network can be placed such that the scaling of total edge length is not allometric.

This definition is similar to the one used in [86] for the fractal dimension of

circuits. From Equation 7.25, the scaling of total wire length is not allometric when:

$$\begin{aligned} \frac{1}{\Delta_l} + \frac{1}{\Delta_w} &\leq 1 \\ \frac{1}{\Delta_l} &\leq 1 - \frac{1}{\Delta_w} \\ \Delta_l &\geq \frac{1}{1 - \frac{1}{\Delta_w}} \\ \Delta_l &\geq \frac{\Delta_w}{\Delta_w - 1}. \end{aligned} \tag{7.31}$$

From Definition 1, the fractal dimension Δ_g corresponds to the lowest value of the spatial dimension Δ_l such that the above condition is satisfied. Therefore,

$$\Delta_g = \frac{\Delta_w}{\Delta_w - 1}, \tag{7.32}$$

or

$$\Delta_g = \frac{1}{1-p}. \tag{7.33}$$

This result proves the relation between p and Δ_g, which is identical to the relation previously suggested by Stroobandt (2001) [86] but for which no proof was presented.

7.4 Energy-delay product

The energy-delay product is a widely accepted metric of cost in computer architectures [33, 1]. This metric implies that, to be efficient, computer designs must minimize cost by managing the trade-off between energy consumption and performance. In this section, we derive a general equation for the scaling of energy-delay product and analyze the conditions leading to its minimization.

This section is organized as follows. From first principles, we provide general expressions for the scaling of resistance, capacitance, latency, and bandwidth. These

expressions are then used to compute energy, delay, and the energy-delay product. All derivations use the most basic formulas and all the asymptotic approximations used are justified in the text. Next, we determine the conditions for which the energy-delay product is optimized. The final result is unexpected: for optimal energy-delay product, all three scaling dimensions must be the same.

As a guide, Table 7.2 contains a summary of the variables introduced in this section.

Table 7.2: List of the variables introduced in Section 7.4.

Variable	Description
R	Resistance
C	Capacitance
L	Latency
B	Bandwidth
E	Energy
D	Delay
O	Output

7.4.1 Resistance

Ohm's law states that the resistance of a conductor is proportional to its length divided by its cross-sectional area as

$$R = \frac{\rho l}{A}, \tag{7.34}$$

where ρ is the resistivity of the material. Although there have been improvements in material resistivity of wires (for example, using copper wires instead of aluminum), ρ has scaled very slowly with N [41], so we approximate it to a constant [27]. There is also some variation in the aspect ratio of wires, which could affect the scaling of area. However, this variation is very small relative to l [93] and, therefore, we assume

wires have a fixed aspect ratio and that area is proportional to r^2, as discussed in Section 7.2.2. As a result, we obtain the following scaling equation for resistance:

$$R \propto \frac{l}{r^2}. \tag{7.35}$$

7.4.2 Capacitance

For wires with fixed aspect ratio and negligible fringing effects, the scaling of capacitance is given as [93]:

$$C \propto \epsilon l, \tag{7.36}$$

where ϵ is the dielectric constant. Although there have been improvements in the dielectric constant of materials, ϵ has also scaled very slowly with N [41], so we approximate it to a constant. As a result, the simplest form for the scaling of wire capacitance is

$$C \propto l. \tag{7.37}$$

7.4.3 Latency

Wire delay, or latency, is given by the time constant RC [93]. Thus, from the product of Equations 7.35 and 7.37, we obtain the scaling of latency as:

$$L = RC \propto \frac{l^2}{r^2}. \tag{7.38}$$

For hierarchical level i, wire latency can be written as

$$\begin{aligned} L_i &\propto \frac{l_0^2 b^{\frac{2i}{\Delta_l}}}{r_0^2 b^{\frac{2i}{\Delta_r}}} = \frac{l_0^2}{r_0^2} b^{2i\left(\frac{1}{\Delta_l}-\frac{1}{\Delta_r}\right)} \\ L_i &= L_0 \cdot b^{2i\left(\frac{1}{\Delta_l}-\frac{1}{\Delta_r}\right)}, \end{aligned} \tag{7.39}$$

where L_0 is the latency of the smallest wire.

7.4.4 Bandwidth

In general, for a system of spatial dimension Δ_l, bandwidth scales as the space with dimension $\Delta_l - 1$. For example, in a three-dimensional system, bandwidth scales as surface area; in a two-dimensional system, bandwidth scales as length, and so forth. From the generalized volume-area relationship of fractals [65, 86], we write the scaling of bandwidth as

$$B \propto N^{\frac{\Delta_l - 1}{\Delta_l}} = N^{1 - \frac{1}{\Delta_l}}. \tag{7.40}$$

Therefore, for a 1D ring topology, bandwidth is constant with the number of nodes. For a 2D mesh or torus topology, bandwidth scales as $N^{\frac{1}{2}}$, and for a 3D mesh or torus, bandwidth scales as $N^{\frac{2}{3}}$ [32]. Equation 7.40 generalizes this notion for non-integer dimensions.

For hierarchical level i, the network bandwidth can be written as

$$B_i = B_0 \cdot b^{i\left(1 - \frac{1}{\Delta_l}\right)}, \tag{7.41}$$

where B_0 is the bandwidth for a network with one node.

7.4.5 Energy

The energy consumption of a wire is given as follows [89]:

$$E = \frac{CV^2}{2}, \tag{7.42}$$

according to which energy depends on the square of the voltage, V. However, in the history of microprocessor evolution, while N has scaled by a factor of 10^6, V has decreased only by a factor of 10 [71], which compared to V^2 is a difference of

four orders of magnitude. Additionally, fundamental constraints imposed by signal reliability and noise issues currently limit the further scaling of voltage [41]. Like others [53, 89], we assume that C scales much faster than V and that the scaling of energy is given as

$$E \propto C. \tag{7.43}$$

From this simple expression, the total network energy is thus obtained by summing the capacitance of all wires, which is equivalent to the total wire length given in Equation 7.25. Considering both allometric and isometric scaling, we can write the scaling of energy consumption as

$$E \propto \begin{cases} N^{\frac{1}{\Delta_l}+\frac{1}{\Delta_w}}, \text{ for } \frac{1}{\Delta_l}+\frac{1}{\Delta_w} > 1 \\ N, \text{ for } \frac{1}{\Delta_l}+\frac{1}{\Delta_w} < 1. \end{cases} \tag{7.44}$$

Notice that, because the energy used for computation scales as N, it does not affect the scaling of total energy and is ignored in this analysis. It is indeed the case in modern chips that wire capacitances dominate gate capacitances [42].

7.4.6 Delay

The communication delay (or communication overhead) is given by the sum of the transmission delay and latency as [96]:

$$D = \frac{W}{B} + L, \tag{7.45}$$

where W is the amount of communication. From Equations 7.17, 7.39 and 7.41, the delay at hierarchical level i can be written as

$$\begin{aligned} D_i &= \frac{w_0 \cdot b^{\frac{i}{\Delta_w}}}{B_0 \cdot b^{i\left(1-\frac{1}{\Delta_l}\right)}} + L_0 b^{2i\left(\frac{1}{\Delta_l}-\frac{1}{\Delta_r}\right)} \\ &= \frac{w_0}{B_0} b^{i\left(\frac{1}{\Delta_l}+\frac{1}{\Delta_w}-1\right)} + L_0 b^{2i\left(\frac{1}{\Delta_l}-\frac{1}{\Delta_r}\right)}. \end{aligned} \tag{7.46}$$

The total network delay is obtained by summing the communication delays at each hierarchical level, which yields

$$D = \frac{w_0}{B_0} \sum_{i=0}^{H-1} b^{i\left(\frac{1}{\Delta_l}+\frac{1}{\Delta_w}-1\right)} + \frac{l_0^2}{r_0^2} \sum_{i=0}^{H-1} b^{2i\left(\frac{1}{\Delta_l}-\frac{1}{\Delta_r}\right)}. \tag{7.47}$$

From Equation 7.47 above, considering both allometric and isometric scaling, we can write the scaling of network delay as

$$D \propto \begin{cases} N^{\frac{1}{\Delta_l}+\frac{1}{\Delta_w}-1} + N^{2\left(\frac{1}{\Delta_l}-\frac{1}{\Delta_r}\right)}, \text{ for } \frac{1}{\Delta_l}+\frac{1}{\Delta_w} > 1 \text{ and } \Delta_r > \Delta_l \\ 1, \text{ for } \frac{1}{\Delta_l}+\frac{1}{\Delta_w} < 1 \text{ and } \Delta_r < \Delta_l. \end{cases} \tag{7.48}$$

Notice that, because computation delay is constant with the size of the system, it does not affect the scaling of total delay and is ignored in this analysis. This is in accordance with what is observed in today's chips, in which wire delays are much higher than gate delays [42].

7.4.7 Energy × Delay

We now propose a general equation for the scaling of the average energy-delay product and analyze the conditions for which it is minimized. From Equations 7.44 and 7.48, we can write the scaling of total energy-delay product as

$$E \times D \propto \begin{cases} N^{\frac{1}{\Delta_l}+\frac{1}{\Delta_w}} \times \left(N^{\frac{1}{\Delta_l}+\frac{1}{\Delta_w}-1} + N^{2\left(\frac{1}{\Delta_l}-\frac{1}{\Delta_r}\right)} \right), \text{ allometric} \\ N, \text{ isometric,} \end{cases} \tag{7.49}$$

where the allometric conditions are $\frac{1}{\Delta_l}+\frac{1}{\Delta_w} > 1$ and $\Delta_r > \Delta_l$. The average energy-delay product, or average cost per output, is the total energy-delay divided by the total output. The total output is proportional to the external communication of the network, thus:

$$\overline{E \times D} = \frac{E \times D}{O} = \frac{E \times D}{N^{\frac{1}{\Delta_w}}}. \tag{7.50}$$

As a result,

$$\overline{E \times D} \propto \begin{cases} N^{\frac{1}{\Delta_l}} \times \left(N^{\frac{1}{\Delta_l}+\frac{1}{\Delta_w}-1} + N^{2\left(\frac{1}{\Delta_l}-\frac{1}{\Delta_r}\right)} \right), \text{ allometric} \\ N^{1-\frac{1}{\Delta_w}}, \text{ isometric.} \end{cases} \tag{7.51}$$

Finally, we perform a convenient change in variables, which simplifies the above expression. Instead of using the width dimension Δ_w, we express the average energy-delay product as a function of the fractal dimension of the network Δ_g, where, from Section 7.3.3, $\frac{1}{\Delta_g} = 1 - \frac{1}{\Delta_w}$. Hence,

$$\overline{E \times D} \propto \begin{cases} N^{\frac{1}{\Delta_l}} \times \left(N^{\frac{1}{\Delta_l}-\frac{1}{\Delta_g}} + N^{2\left(\frac{1}{\Delta_l}-\frac{1}{\Delta_r}\right)} \right), \text{ for } \Delta_g > \Delta_l \text{ and } \Delta_r > \Delta_l \\ N^{\frac{1}{\Delta_g}}, \text{ for } \Delta_g < \Delta_l \text{ and } \Delta_r < \Delta_l. \end{cases} \tag{7.52}$$

The above expression defines the scaling of the average energy-delay product as a function of the geometric parameters of the network. From this equation, we now determine the optimal values for those parameters so that cost is minimized.

For the allometric case, the minimum cost is obtained when Δ_g and Δ_r are as low as possible. Since $\Delta_g > \Delta_l$ and $\Delta_r > \Delta_l$, it follows that the minimum cost occurs when Δ_g and Δ_r are infinitely close to Δ_l, or, using standard calculus notation, $\Delta_g \to \Delta_l^+$ and $\Delta_r \to \Delta_l^+$. Intuitively, this corresponds to the case in which bandwidth and communication scale at the same pace and latency is constant with distance, so that there is no slow-down with scaling and performance is the maximum possible. The formula also indicates that the higher the spatial dimension, the lower the cost, which gives a compelling argument to why developing a 3D chip technology is advantageous.

For the isometric case, cost is minimized as Δ_g increases. Since $\Delta_g < \Delta_l$, the minimum cost is obtained when Δ_g is infinitely close to Δ_l. The formula does not contain Δ_r and, therefore, Δ_r could have any value smaller than Δ_l with no change in the energy-delay. However, the lower the Δ_r the thicker the wires, and since there

are obvious spatial and material costs in making wires thicker, the best value for Δ_r is one infinitely close to Δ_l. We conclude that, for isometric scaling, cost is minimized when $\Delta_g \to \Delta_l^-$ and $\Delta_r \to \Delta_l^-$.

Our analysis shows that the optimal system exists in the limit between allometric and isometric scaling, although never exactly at pseudo-allometric scaling. Nevertheless, for all practical purposes, the optimal design is achieved when:

$$\Delta_g = \Delta_r = \Delta_l. \tag{7.53}$$

This result leads to an extremely simple geometrical model in which three independent scaling dimensions collapse into a single parameter. If a system is optimized for energy-delay product, only one of its geometric dimensions is needed in order to guess the other two. For chips, $\Delta_l = 2$ and, therefore, it should be expected that $\Delta_g \approx 2$ and $\Delta_r \approx 2$. Interestingly, a value of $\Delta_g \approx 2$ implies $p \approx 0.5$ and, in fact, the average Rent's exponent of microprocessors has been found to be approximately 0.45 [4]. For the scaling of wire thickness, the result that $\Delta_r = \Delta_l$ corresponds to the case in which thickness and length scale at the same rate. This matches exactly with the prediction by Dennard's scaling theory for the ideal scaling of wires [27].

7.5 Power and performance

From fundamental electrical and geometrical principles, we have derived a general expression for the scaling of average energy-delay product, and used this expression to determine the geometry of an ideal computing system that has minimum cost. We postulate that if real computing systems are designed to optimize the same metric, then the scaling of real-world computers should approach to a certain degree the scaling of the ideal system.

In this section, we assume optimal network scaling and use the fact that $\Delta_g = \Delta_r = \Delta_l$ to derive general scaling relations for power and performance in computing systems. We show that these scaling relations predict with high accuracy the scaling of power and throughput of real-world systems across a range of several orders of magnitude.

7.5.1 Power

With the assumption of optimal scaling the computation of power consumption is simple. Power is the energy consumed per unit of time, or energy divided by delay:

$$P = \frac{E}{D}. \tag{7.54}$$

From Equation 7.44, energy is given by $E \propto N^{\frac{1}{\Delta_l}+\frac{1}{\Delta_w}}$, and from Equation 7.32, $\frac{1}{\Delta_w} = 1 - \frac{1}{\Delta_g}$. Thus,

$$E \propto N^{\frac{1}{\Delta_l}+\frac{1}{\Delta_w}} = N^{1+\frac{1}{\Delta_l}-\frac{1}{\Delta_g}}. \tag{7.55}$$

Since in optimal scaling $\Delta_g = \Delta_l$, it results that $E \propto N$. In the case of delay, under optimal scaling Equation 7.48 leads to $D \propto 1$. Therefore, the scaling of power is given as

$$P \propto N. \tag{7.56}$$

There is only one component missing in the above formulation, which is feature size reduction. We did not introduce this factor before because it has no influence in the results obtained so far and, therefore, was not relevant to the previous discussion. However, in order to compute power consumption, the shrinkage of device dimensions must be accounted for. In real systems, the chip area is approximately constant while the number of transistors increases, so that the length of wires decreases as $N^{\frac{1}{\Delta_l}}$ [70]. This has no impact on delay, which is independent of distance in the optimal

scenario, but has a proportional effect on energy. Therefore, the final equation for power consumption is given as:

$$P \propto \frac{N}{N^{\frac{1}{\Delta_l}}} = N^{1-\frac{1}{\Delta_l}}. \tag{7.57}$$

For two-dimensional chips, $\Delta_l = 2$, thus

$$P \propto N^{\frac{1}{2}}. \tag{7.58}$$

We compare this prediction with data obtained for 523 different microprocessors over a range of approximately 6 orders of magnitude. Figure 7.6 shows the scaling of power consumption for the real data, where the slope obtained with a linear regression is 0.495, which agrees very closely with the prediction of 0.5. The correlation coefficient between observed and predicted power is 0.81.

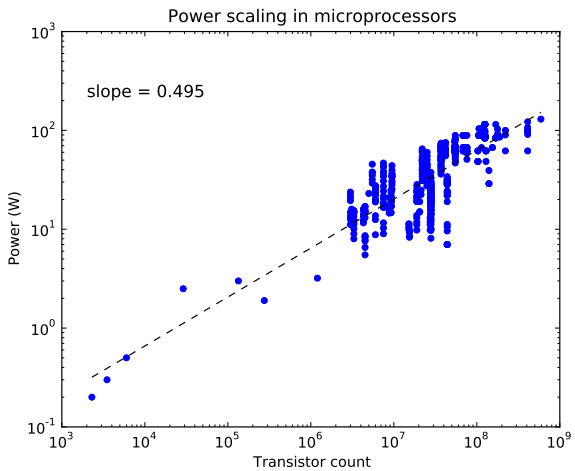


Figure 7.6: The scaling of power consumption as a function of the number of transistors for 523 microprocessors of different vendors and technological generations.

This result provides supporting evidence to our hypothesis that computer systems, through careful design optimization, do indeed approach the optimal or ideal design in terms of energy-delay product minimization. Our model provides a simple explanation for the scaling of power consumption in computer architectures over 40 years of history of computer technology.

7.5.2 Performance

Performance is usually measured as throughput, i.e., the number of instructions executed per unit of time. Under the assumption of optimal scaling it is also easy to compute throughput, which is given as

$$T \propto \frac{N}{D}. \tag{7.59}$$

Since $D \propto 1$, we predict that throughput scales linearly with size:

$$T \propto N. \tag{7.60}$$

It is difficult to obtain consistent performance data for microprocessors because there is no standard. The original metric was MIPS (million instructions per second), but this metric was dropped many years ago and vendors have defined their own metrics, which also have changed over time. We were able to obtain normalized performance data for 16 Intel chips, ranging from the first microprocessor to be produced, the Intel 4004 from 1971, to the modern Intel Quad Core Xeon from 2007. Although this dataset is much smaller than the dataset for power consumption, it uniformly covers a range of 6 orders of magnitude, which is a more desirable trait than simply sample size when measuring power-laws.

Figure 7.7 shows the scaling of normalized throughput for the 16 Intel processors which, despite some variance, displays a consistent linear trend. A linear regression shows a slope of 1.0, which is exactly what is predicted by the theory. The correlation coefficient between observed and predicted throughput is 0.97.

The results in this section verify the ability of the proposed theory to explain trends in the evolution of computer technology which, until now, were empirical observations with no theoretical support. Our geometrical framework derived from first principles shows that, driven by the optimization of a single constraint, the scaling of microprocessor systems is governed by extremely simple laws.

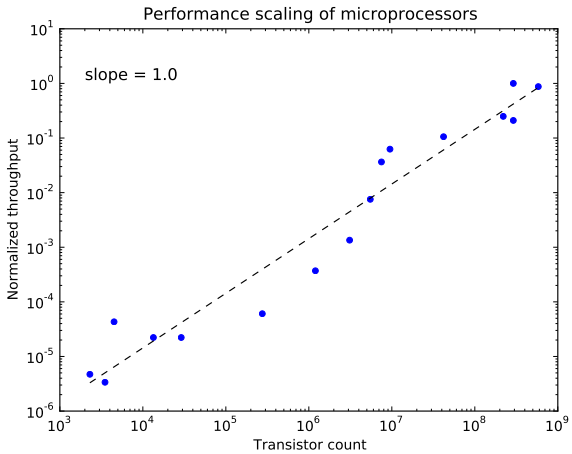


Figure 7.7: The scaling of throughput as a function of the number of transistors for 16 Intel microprocessors of different technological generations.

7.6 Discussion

Consistency of the theory. From a simple model of network scaling, we derived the ideal dimensions of a computing system that has minimum cost. However, real computer architectures are far from being simple: modern microprocessors are composed of billions of transistors arranged in a successive collection of incredibly complex circuits and diverse functionalities. They are also not ideal either: their behavior is affected by many thermal, material, and electrical issues that were not considered here, and their design is largely based on heuristics with no guarantee of optimality.

Nevertheless, the proposed theory is highly consistent with empirical observations. The simple model predicts the scaling exponent of power and throughput with high accuracy. It also correctly predicts that the Rent's exponent of general purpose microprocessors is close to 0.5 (Section 7.4.7). Our framework is also consistent with a number theoretical results obtained independently by other authors and in different areas. From our general model of network scaling, we were able to

reproduce the 3/4 power scaling of vascular networks [92] (Section 7.3.1); the average wire length predicted by Donath [30] (Section 7.3.2); the relationship between the Rent's exponent and the fractal dimension of a network, derived by Stroobandt [86] (Section 7.3.3); and the ideal wire scaling from Dennard's scaling theory [27] (Section 7.4.7). Our theory puts all these results together into a unified, coherent framework.

Clock frequency. An interesting characteristic of the proposed theory is that the results are independent of clock frequency, which makes the analysis of power and performance much simpler. This is paradoxical because clock frequency is known to be a major contributor to computer performance and power consumption. The reason for this is that frequency only affects computation: increasing switching speeds does not make wires run any faster. Since in the limit of a large system communication dominates delay, the pace at which the system runs is ultimately limited by the interconnect.

A consequence of increased clock speeds is that the computation efficiency of the architecture has decreased over time as each node computes proportionally less per clock cycle, as shown in Figure 7.8. In the current scenario where communication delay is much higher than computation delay, frequency can no longer increase and, in order to keep the same computing capacity, the solution is to increase parallelism [64] (as it is already happening with the multi-core architecture). In the future, we expect clock speeds and transistor utilization to stay constant, while throughput continues to increase linearly with size.

Parallel processing. In monolithic architectures the Rent's exponent is defined by the interconnect. However, in parallel architectures, such as the multi-core, the Rent's exponent defined by the communication pattern of parallel applications does not need to be the same as that of the system. Therefore, when running high-

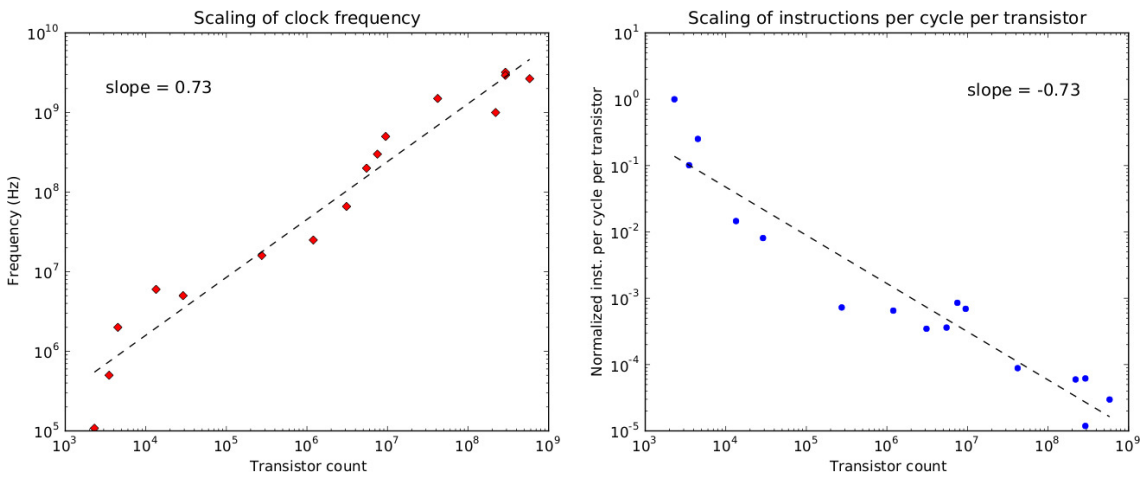


Figure 7.8: The scaling of frequency and of instruction per cycle per transistor. As frequency has increased, each transistor computed proportionally less per cycle.

dimensional applications, such that $\Delta_g > \Delta_l$, the power-performance characteristics will be different than expected, since Δ_g is no longer optimal. In this case, the application performance is expected to slow down and scale as

$$T \propto N^{\frac{1}{\Delta_l} - \frac{1}{\Delta_g}}, \tag{7.61}$$

in particular because the bandwidth of the system is smaller than the required bandwidth to run the application properly. Since the system runs slower, the power consumption will also decrease, scaling as

$$P \propto N^{1 - \frac{1}{\Delta_g}}. \tag{7.62}$$

For applications that have very low dimensionality such that $\Delta_g < \Delta_l$ no slowdown is expected, since bandwidth is not a bottleneck. However, such applications will be operating at a sub-optimal region in terms of energy-delay product, since the average energy consumption does not change as Δ_g decreases.

3D technology. What is the most efficient computer that can be built? Equation 7.52 shows that the higher the Δ_l the lower the average energy-delay product. As

a result, the most efficient computer has the highest possible spatial dimension, which is 3D. The development of a 3D integrated circuit technology would, therefore, represent major breakthrough towards the production of the most complex and most efficient computer architectures. Interestingly, the fractal geometry of the folded cortical surface in the brain is estimated to be in the order of 2.8 [52]. We do not know yet whether the energy-delay analysis presented here would also apply to the brain, but the higher dimensionality of the brain suggests that mammalian brains are much more complex and efficient computing devices than current 2D computer chip architectures, and are close to the limit of maximum efficiency.

7.7 Conclusion

Complexity Theory in computer science analyzes the scaling behavior of algorithms and is widely used in design and implementation of efficient computer software. In contrast, the development of computer hardware is mostly an empirical practice, in which trial-and-error and technical experience play a major role. In computer science, there is no equivalent theory for the scaling of hardware that could guide the design of efficient computing devices and systems.

Employing an interdisciplinary approach, this chapter proposed a theory for the scaling of power and performance in computing. Using a similar geometric framework as metabolic scaling theory in biology, we analyzed the efficiency of computer implementations as a function of the geometry of interconnection networks. Our theory, derived from first principles and based on the most fundamental electrical properties of materials, accurately predicts the scaling of power and performance in microprocessors.

Chapter 8

Conclusions

The computer architecture community has adopted the multi-core design as an attempt to avoid the power wall and further scale the performance of computers. The multi-core architecture is more scalable than the traditional monolithic design, because increasing performance through parallelism consumes less power than increasing clock frequency. However, many challenges still need to be overcome in order to take full advantage of this massive on-chip parallelism. In particular, understanding the impact of different design choices on power and energy consumption is of ultimate importance to the success of multi- and many-core chips in the future.

This dissertation analyzed multiple aspects affecting the scalability of the multi-core design with focus on energy and power consumption on the network on chip. In Chapter 3, we studied the effect of different NoC topologies to the power and performance of multi-core chips as a function of the number of cores. In Chapter 4, we looked at the impact of communication locality of different traffic patterns on energy of the interconnect. Chapter 5 proposed a new method for data placement optimization which greatly reduces the energy consumption used for communication in parallel applications. In Chapter 6, we used Rent's rule, a technique from VLSI

design, to determine a theoretical lower-bound on NoC energy consumption. And in Chapter 7, we devised a theoretical framework that explains observed power-performance trends in the evolution of computer architectures and allows us to look into the future of computer designs.

Appendices

Appendix A

Derivation of CPD for Arbitrary Traffic Patterns

In this appendix, we derive a summation series that enumerates all paths with a certain length on a squared mesh. This is then used to compute the Communication Probability Distribution for any traffic pattern described as the probability of communication between two nodes with distance.

The derivation assumes a 4×4 mesh and then generalizes the results to an $N \times N$ mesh, where N is the number of nodes on one side of the squared mesh. We assume the xy-routing algorithm, i.e., every packet is routed on the x-dimension first and then on the y-dimension. Because paths defined this way have a four-way symmetry (right-down, left-down, right-up, left-up), we only compute the number of paths going in one direction (e.g., right-down) and multiply the result by four.

Figure A.1 all possible paths with length 1 to 6 for the 4×4 mesh using the right-down direction. Associated to each path is the number of times that path occurs as the product between how many times it is repeated in the x and y dimensions. For example, there is only one possible path of length 1, which is repeated 4 times in the

x-dimension and three times in the y-dimension, thus occurring 12 times.

Below we show the generalized forms of the number of paths of a certain length according to the figure:

$$[l = 1] \quad N(N-1) + 0 + 0$$

$$[l = 2] \quad (N-1)(N-1) + N(N-1) + 0$$

$$[l = 3] \quad (N-2)(N-1) + (N-1)(N-2) + N(N-3)$$

$$[l = 4] \quad (N-3)(N-1) + (N-2)(N-2) + (N-1)(N-3)$$

$$[l = 5] \quad 0 + (N-3)(N-2) + (N-2)(N-3)$$

$$[l = 6] \quad 0 + 0 + (N-3)(N-3)$$

From the above formulas we deduce a generalized summation series for the number of paths with length l for an arbitrarily sized mesh as:

$$Paths(l) = 4 \cdot \sum_{i=1}^{N-1} (N-i)(N+i-l), \tag{A.1}$$

for $0 < (N + i - l) \leq N$.

The Communication Probability Distribution is then given as:

$$CPD(l) = \frac{P(l) \times Paths(l)}{\sum_{j=1}^{2N-2} Paths(j)}, \tag{A.2}$$

where $P(l)$ is the probability of communication between two nodes with distance l apart.

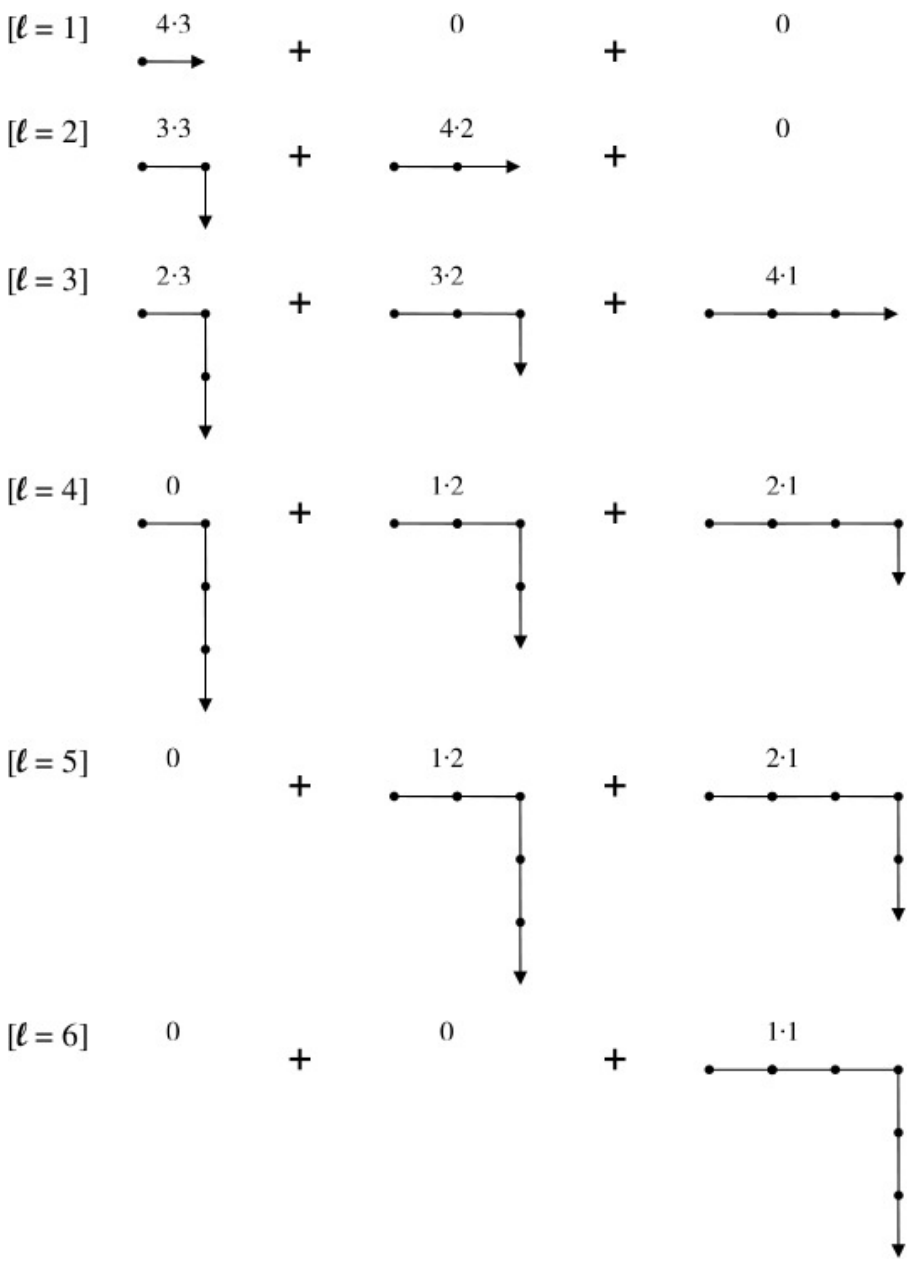


Figure A.1: All possible paths with length l for a 4×4 mesh.

Appendix B

Proof of Total Unimodularity

In this appendix, we prove that the constraint matrix of the integer programming model described in Section 5.4.2 is totally unimodular. The definitions of unimodular and totally unimodular matrices are given as:

Definition 1. *A matrix A is unimodular if it is a square integer matrix with determinant +1 or –1;*

Definition 2. *A matrix A is totally unimodular if every square non-singular submatrix of A is unimodular.*

The following theorem states four conditions that are sufficient for a matrix to be totally unimodular [**?**].

Theorem 1. *Let A be an m by n matrix whose rows can be partitioned into two disjoint sets B and C. The matrix A is totally unimodular if it satisfies the following conditions:*

(a) *Every column of A contains at most two non-zero entries;*

(b) *Every entry in A is 0, +1, or –1;*

(c) *If two non-zero entries in a column of A have the same sign, then the row of one is in B, and the other in C;*

(d) *If two non-zero entries in a column of A have opposite signs, then the rows of both are in B, or both in C.*

The next theorem lists the properties of totally unimodular matrices that will be used in our proof [**?**]:

Theorem 2. *If A is a totally unimodular matrix, the following properties are true:*

(a) *A matrix obtained by duplicating a row or column of A is totally unimodular;*

(b) *A matrix obtained by multiplying a row or column of A by –1 is totally unimodular;*

(c) *The concatenation of A with the identity matrix is totally unimodular.*

Proof. The constraints of the ILP model of Section 5.4.2 are defined in the standard form as

$$A\mathbf{x} \leq \mathbf{b}.$$

The constraint matrix A is given by

$$A = \begin{bmatrix} A_1 \\ A_2 \\ -A_1 \\ A_3 \end{bmatrix},$$

where

$$A_1 = \left[\begin{array}{cccc:cccc:c:cccc} 1 & 1 & \cdots & 1 & 0 & 0 & \cdots & 0 & & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 1 & 1 & \cdots & 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & & 1 & 1 & \cdots & 1 \end{array}\right]$$

.

$$A_2 = \left[\begin{array}{cccc:cccc:c:cccc} 1 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & & 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & 1 & \cdots & 0 & \cdots & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 1 & & 0 & 0 & \cdots & 1 \end{array}\right]$$

$$A_3 = -I,$$

and the vectors $\mathbf{x}$ and $\mathbf{b}$ are written as

$$\mathbf{x} = \left[\begin{array}{cccc:cccc:c:cccc} \pi_{00} & \pi_{01} & \cdots & \pi_{0N} & \pi_{10} & \pi_{11} & \cdots & \pi_{1N} & \cdots & \pi_{B0} & \cdots & \pi_{BN} \end{array}\right]^T$$

$$\mathbf{b} = \left[\begin{array}{cccc:cccc:cccc:cccc} 1 & 1 & \cdots & 1 & K & K & \cdots & K & -1 & -1 & \cdots & -1 & 0 & 0 & \cdots & 0 \end{array}\right]^T.$$

From Theorem 1, it follows that the submatrix of A formed by A_1 and A_2 is totally unimodular, since A_1 and A_2 form two disjoint sets of rows for which the four conditions apply. To show that the entire matrix A is total unimodular, we will use the properties of totally unimodular matrix listed in Theorem 2.

Using properties (a) and (b) of Theorem 2, we take the submatrix formed by A_1 and A_2, duplicate all the rows of A_1 and negate them, creating $-A_1$. This new matrix, formed by A_1, A_2, and $-A_1$ is also totally unimodular. The identity matrix can be appended to this new matrix by applying property (c) of Theorem 2. We then use property (b) to create A_3, thus obtaining the entire A matrix. Therefore, the constraint matrix A is totally unimodular.

Appendix C

The Fourth Scaling Dimension

In this section, we describe how hierarchical model of network scaling accounts for a power-law degree distribution in a way that is independent of geometry of the network.

In complex networks, a power-law degree distribution is known to arise from a "rich gets richer" principle. The canonical explanation for this phenomenon is called *Preferential Attachment* (PA) [7], in which the network grows by adding new nodes that are connected to existing ones with probability proportional to the existing node's degree. We propose an analogous mechanism for the growth of trees, called *Preferential Growth* (PG). This mechanism biases growth towards the bigger branches, leading to a power-law distribution of node sizes and, consequently, degree distribution. We explain this mechanism as follows.

A tree grows recursively as leaf nodes are replaced with b new nodes and branches. If branching is unbiased, the next growing node is chosen randomly from all the leaves. Another possibility is to have all leaf nodes branch at the same time. In PG, the next growing node is chosen with probability proportional to the number of leaves (or size) of each subtree. Starting at the root node, at each branching point

the probability of choosing a child node is proportional to the size of the subtree rooted at that node. This process is applied recursively while the tree is traversed until a leaf node is chosen.

As an example, we used this process to grow a binary tree with 100,000 nodes and measured, for each hierarchical level, the distribution of node sizes (i.e., the number of leaves in the subtree rooted at that node). The result is a power-law distribution with an exponent of approximately -2. Figure C.1 shows the node size distribution at depth 20 from the root of the tree.

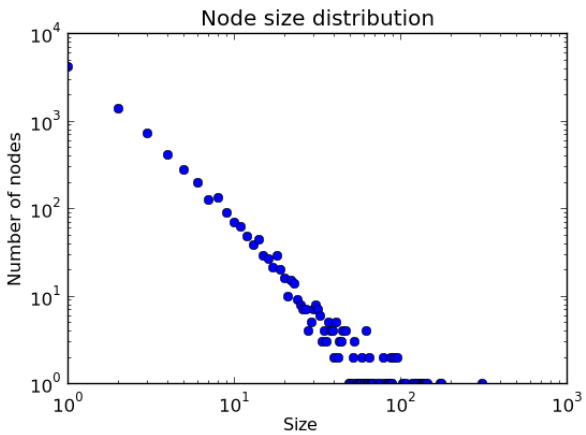


Figure C.1: Power-law distribution of node sizes for a binary tree with 100,000 nodes and depth 20.

We now analyze this result and show how a power-law size distribution leads to a power-law degree distribution. Using the same hierarchical interpretation of fractal dimension, the size of a node at hierarchical level i is

$$s_i = s_0 \cdot b^{i/\Delta_s}, \tag{C.1}$$

where Δ_s is the node size dimension. The frequency of nodes at hierarchical level i is given as

$$f_i = f_0 \cdot b^{-i}. \tag{C.2}$$

Therefore, the frequency of nodes of size s_i can be computed from Equations C.1 and C.2 in the following manner:

$$\begin{aligned} b^i &= (1/s_0^{\Delta_s}) \cdot s_i^{\Delta_s} \\ f(s_i) &= f_0 \cdot \left((1/s_0^{\Delta_s}) \cdot s_i^{\Delta_s}\right)^{-1} \\ f(s_i) &= (f_0 \cdot s_0^{\Delta_s}) \cdot s_i^{-\Delta_s}. \end{aligned} \tag{C.3}$$

In a continuous form, the distribution of sizes is given as

$$f(s) \propto s^{-\Delta_s}. \tag{C.4}$$

Finally, the degree of a node of size s is given by Rent's rule as $k \propto s^p$, or $k \propto s^{\frac{1}{\Delta_w}}$, so the degree distribution can be written as

$$\begin{aligned} f(k) &\propto \left(k^{\Delta_w}\right)^{-\Delta_s} \\ f(k) &\propto k^{-\Delta_w \Delta_s}, \end{aligned} \tag{C.5}$$

which is a power-law.

We conclude that a power-law degree distribution can be accounted for in our model as resulting from a growth process that is independent of the geometry of the network. Notice that the exact value of Δ_s depends on the specific process by which growth occurs.

References

[1] S. Amarasinghe, D. Campbell, W. Carlson, A. Chien, W. Dally, E. Elnohazy, M. Hall, R. Harrison, W. Harrod, K. Hill, et al. Exascale software study: Software challenges in extreme scale systems. *DARPA IPTO, Air Force Research Labs, Tech. Rep*, 2009.

[2] M. Awasthi, K. Sudan, R. Balasubramonian, and J. Carter. Dynamic hardware-assisted software-controlled page placement to manage capacity allocation and sharing within large caches. In *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, pages 250–261, 2009.

[3] F.A.C. Azevedo, L.R.B. Carvalho, L.T. Grinberg, J.M. Farfel, R.E.L. Ferretti, R.E.P. Leite, R. Lent, S. Herculano-Houzel, et al. Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. *The Journal of comparative neurology*, 513(5):532–541, 2009.

[4] H.B. Bakoglu. *Circuits, Interconnections, and Packaging for VLSI*. 1990.

[5] J.R. Banavar, A. Maritan, A. Rinaldo, et al. Size and form in efficient transportation networks. *Nature*, 399(6732):130–131, 1999.

[6] J.R. Banavar, M.E. Moses, J.H. Brown, J. Damuth, A. Rinaldo, R.M. Sibly, and A. Maritan. A general basis for quarter-power scaling in animals. *Proceedings of the National Academy of Sciences*, 107(36):15816–15820, 2010.

[7] A.L. Barabási and R. Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[8] M. Berkelaar, K. Eikland, and P. Notebaert. lp_solve version 5.5. *Eindhoven University of Technology, Design Automation Section, Eindhoven, The Netherlands, ftp://ftp. es. ele. tue. nl/pub/lp solve*, 2006.

[9] B.D. Bingham and M.R. Greenstreet. Computation with energy-time trade-offs: Models, algorithms and lower-bounds. In *Parallel and Distributed Processing with Applications, 2008. ISPA'08. International Symposium on*, pages 143–152. IEEE, 2008.

[10] L. Boroni, Concer, N., Miltos, G., M. Coppola, and R. Locatelli. Noc topologies exploration based on mapping and simulation models. In *10th Euromicro Conference on Digital System Design Architectures*, pages 543–546, 2007.

[11] L. Boroni and N. Concer. Simulation and analysis of network on chip architectures: Ring, spidergon and 2d mesh. In *DATE*, pages 154–159, 2006.

[12] J.A. Brown, R. Kumar, and D. Tullsen. Proximity-aware directory-based coherence for multi-core processor architectures. In *Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, pages 126–134, 2007.

[13] B. Calder, C. Krintz, S. John, and T. Austin. Cache-conscious data placement. *ACM SIGPLAN Notices*, 33(11):139–149, 1998.

[14] E.L. Charnov. Life history invariants: some explorations of symmetry in evolutionary ecology. *booksgooglecom*, 1993.

[15] M. Chaudhuri. PageNUCA: Selected policies for page-grain locality management in large shared chip-multiprocessor caches. In *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, pages 227–238. IEEE, 2009.

[16] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *Wireless Networks*, 8(5):481–494, 2002.

[17] G. Chen, F. Li, and M. Kandemir. Compiler-directed application mapping for noc based chip multiprocessors. In *Proceedings of the 2007 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems*, page 157, 2007.

[18] S. Chen, P.B. Gibbons, M. Kozuch, V. Liaskovitis, A. Ailamaki, G.E. Blelloch, B. Falsafi, L. Fix, N. Hardavellas, T.C. Mowry, et al. Scheduling threads for constructive cache sharing on CMPs. In *Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, page 115, 2007.

[19] Z. Chishti, M.D. Powell, and TN Vijaykumar. Distance associativity for high-performance energy-efficient non-uniform cache architectures. 2003.

[20] Z. Chishti, M.D. Powell, and TN Vijaykumar. Optimizing replication, communication, and capacity allocation in CMPs. In *Computer Architecture, 2005. ISCA'05. Proceedings. 32nd International Symposium on*, pages 357–368. IEEE, 2005.

[21] S. Cho and L. Jin. Managing distributed, shared l2 caches through os-level page allocation. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 455–468, 2006.

[22] P. Christie and D. Stroobandt. The interpretation and application of rent's rule. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(6):639–648, 2000.

[23] William James Dally and Brian Towles. *Principles and Practices of Interconnection Netwoks*. Morgam Kauffman Publishers, San Francisco, 2004.

[24] W.J. Dally and B Towles. Route packets, not wires: on-chip interconnection networks. In *Design Automation Conference, 2001. Proceedings*, pages 684–689, 2001.

[25] J. A. Davis, V. K. De, and J. D. Meindl. A stochastic wire-length distribution for gigascale integration (GSI) - Part I: Derivation and validation. *IEEE Transactions on Electron Devices*, VOL 45(3):580–589, 1998.

[26] Giovanni de Micheli and Luca Benini. *Networks on Chips*. Morgan Kaufmann, 500 Sansome Street, Suite 400, San Francisco CA 94111, 2006.

[27] R.H. Dennard, F.H. Gaensslen, VL Rideout, E. Bassous, and AR LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *Solid-State Circuits, IEEE Journal of*, 9(5):256–268, 1974.

[28] R.P. Dick, D.L. Rhodes, and W. Wolf. TGFF: task graphs for free. In *Proceedings of the 6th international workshop on Hardware/software codesign*, pages 97–101, 1998.

[29] P.S. Dodds. Optimal form of branching supply and collection networks. *Physical review letters*, 104(4):48702, 2010.

[30] W. Donath. Placement and average interconnection lengths of computer logic. *Circuits and Systems, IEEE Transactions on*, 26(4):272–277, 1979.

[31] W. E. Donath. Wire length distribution for placements on computer logic. *IBM J. Res. and Development*, 25:152–155, 1981.

[32] J. Duato, S. Yalamanchili, and L.M. Ni. *Interconnection networks: An engineering approach*. Morgan Kaufmann, 2003.

[33] R. Gonzalez and M. Horowitz. Energy dissipation in general purpose microprocessors. *Solid-State Circuits, IEEE Journal of*, 31(9):1277–1284, 1996.

[34] S.J. Gould. Allometry and size in ontogeny and phylogeny. *Biological Reviews*, 41(4):587–638, 1966.

[35] D. Greenfield, A. Banerjee, J.-G. Lee, and S. Moore. Implications of Rent's rule for NoC design and its fault-tolerance. In *Proceedings of the First International Symposium on Networks-on-Chip (NOCS'07)*, 2007.

[36] B. Grot and S.W. Keckler. Scalable on-chip interconnect topologies. In *2nd Workshop on chip Multiprocessor Memory Systems and Interconnects*, 2008.

[37] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki. Reactive nuca: near-optimal block placement and replication in distributed caches. In *Proceedings of the 36th annual international symposium on Computer architecture*, pages 184–195, 2009.

[38] W. Heirman, J. Dambre, D. Stroobandt, and J.V. Campenhout. Rent's rule and parallel programs: Characterizing network traffic behavior. In *Proceedings of the 2008 International Workshop on System Level Interconnect Prediction, SLIP'08*, 2008.

[39] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach, 4th edition*. Morgan Kaufmann, 500 Sansome Street, Suite 400, San Francisco, CA 94111, 2006.

[40] J.L. Henning. Spec cpu2000: Measuring cpu performance in the new millennium. *Computer*, 33(7):28–35, 2000.

[41] R. Ho. *On-chip wires: scaling and efficiency*. PhD thesis, Citeseer, 2003.

[42] R. Ho, K.W. Mai, and M.A. Horowitz. The future of wires. *Proceedings of the IEEE*, 89(4):490–504, 2001.

[43] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar. A 5-GHz mesh interconnect for a teraflops processor. *IEEE MICRO*, 27(5):51–61, 2007.

[44] J. Hu and R. Marculescu. Energy-aware mapping for tile-based NOC architectures under performance constraints. In *Proceedings of ASP-Design Automation Conference*, pages 233–239, 2003.

[45] J. Hu and R. Marculescu. Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints. *IEEE Computer Society*, 2004.

[46] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S.W. Keckler. A NUCA substrate for flexible CMP cache sharing. *IEEE transactions on parallel and distributed systems*, pages 1028–1040, 2007.

[47] D.N. Jayasimha, B. Zafar, and Y. Hoskote. On-chip interconnection networks: Why they are different and how to compare them. In *blogs.intel.com*, 2006.

[48] A. Kahng, B. Li, L.S. Peh, and K. Samadi. Orion 2.0: A fast and accurate NOC power and area model for early-stage design space exploration. In *Design, Automation, and Test in Europe*, pages 423–428, 2009.

[49] M. Kandemir and G. Chen. Locality-aware process scheduling for embedded mpsocs. In *Design, Automation and Test in Europe, 2005. Proceedings*, pages 870–875, 2005.

[50] J. Kim, J. Balfour, and W.J. Dally. Flattened butterfly topology for on-chip networks. In *40th IEEE/ACM International Symposium on Microarchitecture (MICRO'07)*, 2007.

[51] J. Kim, W.J. Dally, and D. Abts. Flattened butterfly: A cost-efficient topology for high-radix networks. In *Proceedings of the 34rd International Symposium on Computer Architecture (ISCA'07)*, 2007.

[52] V.G. Kiselev, K.R. Hahn, and D.P. Auer. Is the brain cortex a fractal? *Neuroimage*, 20(3):1765–1774, 2003.

[53] G. Kissin. Measuring energy consumption in vlsi circuits: A foundation. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 99–104. ACM, 1982.

[54] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carson, W. Dally, M. Denneau, P. Franzon, W. Harrod, and K. Hill. Exascale computing study: Technology challenges in achieving exascale systems. 2008.

[55] S. Koohi, Mirza-Aghatabar, and S. M., Hessabi. Evaluation of traffic pattern effect on power consumption in mesh and torus network-on chips. In *International Symposium on Integrated Circuits (ISIC'07)*, 2007.

[56] J.G. Koomey. Estimating total power consumption by servers in the US and the world. 2007.

[57] J.G. Koomey. Worldwide electricity used in data centers. *Environmental Research Letters*, 3:034008, 2008.

[58] J.G. Koomey, S. Berard, M. Sanchez, and H. Wong. Implications of historical trends in the electrical efficiency of computing. *Annals of the History of Computing, IEEE*, 33(3):46–54, 2011.

[59] V.A. Korthikanti and G. Agha. Analysis of Parallel Algorithms for Energy Conservation in Scalable Multicore Architectures. In *2009 International Conference on Parallel Processing*, pages 212–219, 2009.

[60] M. Kreutz, C. Marcon, L. Calazans Carro, and A. N. Susin. Energy and latency evaluation of noc topologies,. In *ISCA 2005*, pages 5866–5869, 2005.

[61] B.S. Landman and R.L. Russo. On a pin versus block relationship for partitions of logic graphs. *IEEE Transactions on Computers,*, C-20(12):1469–1479, 1971.

[62] T. Lei and S. Kumar. A two-step genetic algorithm for mapping task graphs to a network on chip architecture. In *Proceedings of the Euromicro Symposium on Digital System Design (DSD'03)*, 2003.

[63] C.E. Leiserson. Fat-trees- university networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, 34:892–901, 1985.

[64] D. Liu and C. Svensson. Trading speed for low power by choice of supply and threshold voltages. *Solid-State Circuits, IEEE Journal of*, 28(1):10–17, 1993.

[65] B.B. Mandelbrot. *The fractal geometry of nature*. Wh Freeman, 1983.

[66] H. Matsutani, M. Koibuchi, and H. Amano. Performance, cost, and energy evaluation of fat h-tree: A cost-efficient tree-based on-chip network. In *IEEE International Parallel and Distributed Processing Symposium*, 2007.

[67] J. Merino, V. Puente, P. Prieto, and J.Á. Gregorio. Sp-nuca: a cost effective dynamic non-uniform cache architecture. *ACM SIGARCH Computer Architecture News*, 36(2):64–71, 2008.

[68] J.E. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal. Graphite: A distributed parallel simulator for multicores. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pages 1–12, 2010.

[69] M. Mirza-Aghatabar, S. Koohi, S. Hessabi, and M. Pedram. An empirical investigation of mesh and torus noc topologies under different routing algorithms and traffic models. In *10th Euromicro Conference on Digital System Design Architectures*, 2007.

[70] M.E. Moses, S. Forrest, A.L. Davis, M.A. Lodder, and J.H. Brown. Scaling theory for information networks. *Journal of the Royal Society Interface*, 5(29):1469, 2008.

[71] Tak H Ning. A perspective on the theory of MOSFET scaling and its impact. *IEEE Solid State Circuits Newsletter*, 12(1):27–30, 2007.

[72] J.C.S. Palma, C.A.M. Marcon, F.G. Moraes, N.L.V. Calazans, R.A.L. Reis, and A.A. Susin. Mapping embedded systems onto NoCs: the traffic effect on dynamic energy estimation. In *Proceedings of the 18th annual symposium on Integrated circuits and system design*, page 201, 2005.

[73] P.P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh. Effect of traffic localization on energy dissipation in NoC-based interconnect. In *ISCA 2005*, pages 1774–1777, 2005.

[74] P.P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh. Performance evaluation and design trade-offs for network-on-chip interconnect architectures. *IEEE Transaction on Computers*, 54(8), 2005.

[75] R. Pop and S. Kumar. A survey of techniques for mapping and scheduling applications to network on chip systems. *School of Engineering, Jonkoping University, Research Report*.

[76] R. Pop and S. Kumar. Mapping applications to noc platforms with multithreaded processor resources. In *NORCHIP Conference, 2005. 23rd*, pages 36–39, 2005.

[77] C.A. Price and B.J. Enquist. Scaling mass and morphology in leaves: an extension of the wbe model. *Ecology*, 88(5):1132–1141, 2007.

[78] D. Rahmati, A. E. Kiasari, S. Hessabi, and H. Sarbazi-Azad. A performance analysis of wk-recursive and mesh networks for network-on-chips. In *Proceedings of the 24th International Conference on Computer Design (ICCD)*, 2006.

[79] B. Rountree, D.K. Lowenthal, S. Funk, V.W. Freeh, B.R. de Supinski, and M. Schulz. Bounding energy consumption in large-scale mpi programs. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, page 49. ACM, 2007.

[80] S. Saeidi, A. Khademzadeh, and A. Mehran. SMAP: An Intelligent Mapping Tool for Network on Chip. In *International Symposium on Signals, Circuits and Systems, 2007. ISSCS 2007*, 2007.

[81] Jonathan Schaeffer. *High Performance Computing Systems and Applications*. Kluwer Academic Publishers, Norwell, Massachusetts 02061 USA, 1998.

[82] A. Schrijver. *Theory of linear and integer programming*. John Wiley & Sons Inc, 1998.

[83] S. Scott, D. Abts, J. Kim, and Dally W.J. The black widow high-radix clos network. In *Proceedings of the 33rd International Symposium on Computer Architecture (ISCA'06)*, 2006.

[84] J. Shalf. The new landscape of parallel computer architecture. *Journal of Physics: Conferece Series 78*, 2007.

[85] V. Soteriou, H. Wang, and L.S. Peh. A statistical traffic model for on-chip interconnection networks. In *Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'06)*, pages 104–116, 2006.

[86] Dirk Stroobandt. *A Priory Wire Length Estimates for Digital Design*. Kluwer Academic Pulishers, Boston, 2001.

[87] S. Suboh, M. Bakhouya, and T. El-Ghazawi. Simulation and evaluation of on-chip interconnect architectures: 2d mesh, spidergon, and wk-recursive network. In *Proceedings of the First International Symposium on Networks-on-Chip (NOCS'08)*, 2008.

[88] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J.-W. Lee, and W. Lee. The Raw microprocessor: A computational fabric for software circuits and general purpose programs. *IEEE MICRO*, 22(PART 2):25–35, 2002.

[89] A. Tyagi. Energy-time trade-offs in vlsi computation. In *Proceedings of the Ninth Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 301–311. Springer-Verlag, 1989.

[90] H.-S. Wang, X. Zhu, L.-S. Peh, and S. Malik. Orion: A power performance simulator for interconnection networks. In *Proceedings of the 35th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-35)*, 2002.

[91] G.B. West and J.H. Brown. The origin of allometric scaling laws in biology from genomes to ecosystems: towards a quantitative unifying theory of biological structure and organization. *Journal of Experimental Biology*, 208(9):1575–1592, 2005.

[92] G.B. West, J.H. Brown, and B.J. Enquist. A general model for the origin of allometric scaling laws in biology. *Science*, 276(5309):122, 1997.

[93] N.C. Wilhelm. *Why Wire Delays Will No Longer Scale for VLSI Chips*. Sun Microsystems Laboratories, 1995.

[94] W. Wolf. *Modern VLSI design: IP-based design*. Prentice-Hall PTR, 2008.

[95] S.C. Woo, M. Ohara, E. Torrie, J.P. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *Proceedings of the 22nd annual international symposium on Computer architecture*, pages 24–36, 1995.

[96] Z. Xu and K. Hwang. Modeling communication overhead: Mpi and mpl performance on the ibm sp2. *Parallel & Distributed Technology: Systems & Applications, IEEE*, 4(1):9–24, 1996.

[97] M. Zhang and K. Asanovic. Victim replication: Maximizing capacity while hiding wire delay in tiled chip multiprocessors. In *Computer Architecture, 2005. ISCA'05. Proceedings. 32nd International Symposium on*, pages 336–345. IEEE, 2005.