ON-CHIP NETWORK-ENABLED MANY-CORE ARCHITECTURES FOR COMPUTATIONAL BIOLOGY APPLICATIONS

By

TURBO MAJUMDER

A dissertation submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

Washington State University
School of Electrical Engineering and Computer Science

May 2013

To the Faculty of Washington State University:

The members of the Committee appointed to examine the dissertation of TURBO MAJUMDER find it satisfactory and recommend that it be accepted.

_____

Partha P. Pande, Ph.D., Chair

_____

Ananth Kalyanaraman, Ph.D.

_____

José G. Delgado-Frias, Ph.D.

_____

Eric H. Roalson, Ph.D.

ACKNOWLEDGMENT

I am grateful to my Ph.D. advisor, Dr. Partha Pratim Pande, who has guided me in matters related to my research and otherwise. I do not see myself having carried through the long years during my Ph.D. without his constant support. I express my gratitude to Dr. Ananth Kalyanaraman, my doctoral co-advisor, for the insight that he constantly provided me with during our discussions on various topics. I thank Dr. José G. Delgado-Frias and Dr. Eric H. Roalson for agreeing to be on my Ph.D. committee despite their pressing schedules. To all the students at the Low Power and Robust Nanosystems Lab, past and present, thank you, for making these years an enjoyable learning experience. I would also like to thank the staff of School of EECS, and Allen Guyer in particular, for the amount of help that I have received from them with regard to my work. Outside school, my life in Pullman has been enriched by the friends and acquaintances I have made here.

My parents and other family members have been a source of strength and support throughout these years that I spent far away from home. Words cannot express how much I am indebted to them.

Finally, I would like to thank National Science Foundation for the grant (IIS-0916463) that helped support my doctoral thesis research.

ON-CHIP NETWORK-ENABLED MANY-CORE ARCHITECTURES FOR COMPUTATIONAL BIOLOGY APPLICATIONS

Abstract

by Turbo Majumder, Ph.D.
Washington State University
May 2013

Chair: Partha P. Pande

Large-scale integration of multiple cores on a single chip is the current answer to the challenge of attaining higher computation throughput while restricting power consumption within acceptable limits. Network-on-Chip (NoC) is an emerging paradigm that can efficiently support integration of a massive number of cores on a chip by decoupling the on-chip computation and communication infrastructure, thereby overcoming scalability issues faced by conventional buses.

Many scientific computing disciplines, such as computational biology, have seen a significant increase in the availability of parallel algorithms and high-performance computing (HPC) tools owing to high runtime complexities and/or the data-intensive nature underlying the computation. Software-only solutions are likely to be inadequate, creating the need for hardware accelerators. This dissertation explores the design and development of highly optimized NoC-based hardware accelerators for a particular class of biocomputing applications, viz. phylogeny reconstruction, which is important for evolutionary inferences in computational biology.

This dissertation focuses on two computationally distinct phylogeny reconstruction approaches to demonstrate that NoC-based many-core platforms can deliver orders of magnitude reduction in time-to-solution, compared to

existing approaches. The Maximum Parsimony (MP) phylogeny reconstruction problem can be reduced to one of solving numerous instances of the classical Traveling Salesman Problem (TSP). 99% of the total software runtime is spent in computing TSP instances, whose solution typically involves an application of branch-and-bound runtime heuristics. This dissertation presents the design of many-core systems with core-level pipelined micro-parallel architecture and different interconnection topologies to achieve significant speedup and energy efficiency. In Maximum Likelihood (ML) phylogeny reconstruction, the improved quality of result comes at a higher computational cost, as this approach involves optimization over multi-dimensional real continuous space. We present NoC-based hardware accelerators that target function kernels contributing to a bulk of the runtime. These platforms combine novel ideas and approaches, such as space-filling Hilbert curves, parallelized core allocation schemes, and 3-D integration. We also explore the use of long-range on-chip wireless links on existing regular topologies to reduce network diameter, thereby reducing the average communication latency between cores. These platforms have the potential to serve a broader class of throughput-oriented HPC applications.

Table of Contents

List of Figures

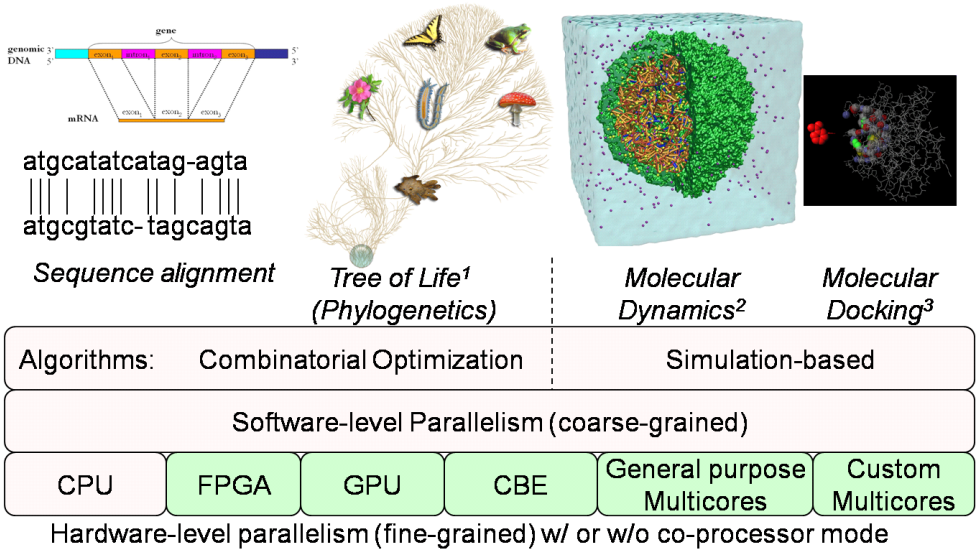
List of Tables

1. Introduction

Computing research has become a vital cog in the machinery required to drive biological discovery. Computing has made possible significant achievements over the last decade, especially in the genomics sector. This has led to immense interest in the field of computational biology. Applications in this field can be classified into those based on combinatorial optimization and those that are simulation-based. The former category includes sequence alignment of genomes and phylogenetic tree reconstruction based on aligned DNA or protein sequences. The latter category includes molecular dynamics and molecular docking. All of these applications are data-intensive or compute-intensive. The usual approach has been to carry out data processing for these applications in software. However, with large amounts of biological data available, the software-only approach has become infeasible owing to inordinately large run-times involved. An emerging area is the investigation of hardware accelerators for speeding up the massive scale of computation needed in such large-scale biocomputing applications. Various hardware platforms, such as Field Programmable Gate Array (FPGA), Graphics Processing Unit (GPU), Cell Broadband Engine (CBE) and multi-core processors are being explored. Figure 1.1 summarizes the current state of the art.

The target of this thesis is the design and evaluation of multicore-based hardware accelerators for phylogenetics. Phylogenetics is the study of evolutionary relationships among organisms based on their underlying genetic content. The term *genome* is a collective reference to all the DNA in the living

cell of an organism and phylogenetic tree construction is the process of building an evolutionary tree based on the similarities and differences observed among the genomic DNA of a set of species. It is a fundamental problem in computational molecular biology with important applications that include drug discovery. In this tree, the leaves represent species (known) and the internal nodes represent common ancestral species (unknown). Until a decade ago, only a handful of genome sequences were available and therefore the knowledge regarding evolutionary trees was limited. However, with the recent advances in



(Image credits (1) http://www.tolweb.org/tree/ (2) http://www.ks.uiuc.edu/Research/STMV/ (3) http://wwwcs.uni-paderborn.de/~lst/HotDock/)

Figure 1-1: Biocomputing applications benefiting from hardware acceleration

DNA sequencing technologies, sequence information for more than a thousand species is now available in public databases and more large-scale sequencing efforts are currently underway. Owing to this deluge in genomic information, the computational biology community has embarked on a project called the Tree of Life, which is an ambitious project to construct the evolutionary tree connecting all known species. The single largest impediment to this project is, however, the

high computational costs associated with building phylogenetic trees [1]. Figure 1.2 provides an example of a typical phylogenetic tree.

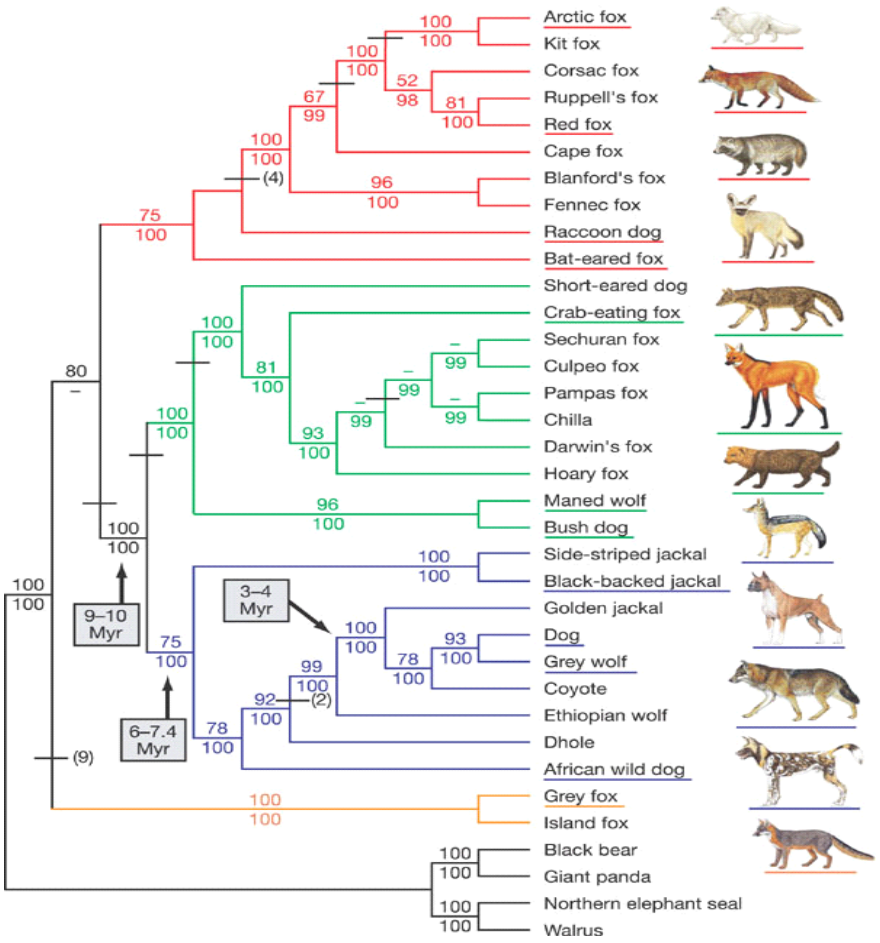


Figure 1-2: Phylogenetic tree showing members of the dog family

Reproduced with permission from K. L. Toh et al, "Genome sequence, comparative analysis and haplotype structure of the domestic dog", Nature 438, 803-819 (8 December 2005)

The process of inferring the phylogeny of a set of *k taxa* (or species) entails reconstructing a phylogenetic tree based on distance or probability measures [2]. Most approaches for phylogenetic tree reconstruction are based on Neighbor Joining (NJ), Maximum Parsimony (MP), Maximum Likelihood (ML) and Bayesian Inference (BI). When the relative ordering of genes on a genome is known, then a specific type of phylogeny called the breakpoint phylogeny can be computed, based on the *breakpoint distance*. Given a reference set of *m* genes

$\{g_1, g_2, \ldots g_m\}$, any genome can be represented by an ordering of the subset of genes that constitute it, as they appear from end to end of the genomic DNA. The *breakpoint distance* between any two genomes is defined as the number of gene pairs that appear adjacent in one genome but not in the other. It is a measure of how different two genomes are by their gene ordering. Blanchette et al. pioneered the work on breakpoint-based phylogeny [3]. They reduced the problem of constructing an optimal phylogenetic tree of N genomes to one of solving numerous instances of a version of the Traveling Salesman Problem (TSP) [4] where edge-weights of the input graph are bounded to a fixed set of integer values. Put intuitively, each instance of TSP tries to identify the gene order of a hypothetical ancestral genome that is the closest representative to any three given genomes. This problem is called the *3-median breakpoint problem* and has been proven to be NP-Hard [5]. This is the primary method used in MP (breakpoint) phylogenetic tree reconstruction.

Probability-based approaches for phylogenetic inference, like ML and BI, provide the most accurate estimate of evolutionary relationship among species. These methods use one of several probabilistic models of evolution, e.g., Jukes-Cantor [6], Kimura-2P [7], HKY85 [8] or General Time Reversible (GTR) [9], [10]. They provide a statistical likelihood score for each reconstructed tree using the Phylogenetic Likelihood Function (PLF) [11], [12]. The boost in quality, however, comes at a high computation cost as both ML and BI formulations are NP-Hard [13] and suffer from the need to explore an super-exponential (in k) number of trees before they come up with an answer. Therefore, they need to rely on algorithmic heuristics and high-performance computing for achieving

practical solutions. The increasing availability of genomic data has only exacerbated the situation.

Methods for phylogenetic tree reconstruction are either data-intensive or computation-intensive, or both. Most approaches for solving such problems rely on the use of parallelism, which divides the problem into *a large number of* smaller semi-independent sub-problems that can be computed *concurrently*. The key points to note here are the requirements of

(a) a large number of computation cores (to deal with each sub-problem or a set of sub-problems), and

(b) effective (i.e. low-latency) communication among cores (to exchange information among semi-independent sub-problems).

Network-on-Chip (NoC) is an emerging paradigm for large scale system integration on a single chip. Instead of the bus-based communication architecture in multi-core System-on-Chips (SoCs), the NoC-based solution proposes a communication infrastructure where various cores exchange data with the help of switches/ routers and links. These platforms mitigate the inter-core communication bottlenecks that appear with larger number of cores on a single chip, thereby enabling integration of more components. Applications involving many computation kernels communicating with one another are typically benefited from a NoC-based platform. Most phylogenetic tree reconstruction applications (e.g., MP, ML, BI, etc.) fall under this category. Hence, NoC-based platforms are a natural choice when attempting to accelerate these applications.

1.1 Contributions

The principal contributions in this dissertation are:

(1) Design and evaluation of NoC-based platform for MP (breakpoint) phylogeny reconstruction, which comprises core and on-chip network design, and benchmarking against existing approaches. We achieved a speedup of over 8430x over multithreaded software, which represents almost an order of magnitude improvement over existing hardware acceleration solutions (~1005x, see Table 1).

(2) Design and evaluation of NoC-based platform for accelerating targeted kernels in ML phylogeny reconstruction, which comprises core and on-chip network design, and design and evaluation of different NoC architectures (including 3D NoC) and job allocation policies. We achieved function-level speedups of up to 847x, and aggregate speedup of the targeted kernels exceeding 6500x over baseline software runs, which represents an order of magnitude improvement over existing hardware acceleration solutions (~381x, see Table 1).

(3) Design and evaluation of NoC-based platforms for throughput-oriented scientific applications, and subsequently using the model to study the effect of using long-range on-chip wireless links in conjunction with different resource allocation strategies on reducing the overall on-chip communication and enhancing computational throughput. We have been able to achieve a computation throughput of 10^{11} operations per second, with each operation consuming ~0.5 nJ.

In the following, we elaborate on the significance of the contributions.

1.1.1 Accelerating Maximum Parsimony Phylogenetic Tree Reconstruction

Our principal contributions here are as follows:

a) We designed a NoC-based platform for computing breakpoint phylogeny by solving multiple instances of TSP using branch-and-bound method. Our architecture provides for efficient run-time and memory management. We have been able to achieve significant speedup over multi-threaded software, up to a high of 8430x for some inputs.
b) We compared two important NoC network topologies – mesh and quad-tree – in terms of run-time and energy performance, and conclusively showed that quad-tree provides better communication latency and energy performance.
c) We designed and evaluated a wide range of synthetic test cases to establish a relationship between properties of input data and performance of our design. We also correlate the performance obtained using real genomic data to these observations. Our experiments show that our platform is able to provide an order of magnitude speedup over existing hardware accelerators, especially when the input genomes are widely disparate.

1.1.1.1 Significance

In the case of MP (breakpoint) phylogenetic tree reconstruction, over 99% of the software run-time is spent in computing instances of TSP [14]. TSP is a widely studied NP-Complete problem for which several heuristics have been

explored [15], [16], [17], [18], [19], [20], [21] and *branch-and-bound* methods [21], [22] continue to be the most popular among accurate solvers, owing to their effectiveness in reducing the super-exponential search space. The run-time heuristic, which itself is computationally intensive, is an ideal candidate for parallelization. An array of processing elements (PEs) working in parallel on distinct parts of the solution would naturally enhance performance. However, these PEs cannot work in isolation and need to communicate amongst themselves. This communication needs to be efficient and synchronized with the computation operation of the PEs. To achieve this in an on-chip scenario, a platform possessing inherent fine-grained, large-scale parallelism and an efficient communication fabric needs to be chosen. It is clear that a NoC-based solution provides the best fit to this requirement. On one hand, a NoC scales very well with increasing number of PEs; on the other hand, it offers the user the freedom to choose the communication architecture that is most apt for a target application. In this perspective, our contributions mentioned above have been able to deliver significantly shorter time-to-solution while being energy-efficient.

1.1.2 Accelerating Maximum Likelihood Phylogenetic Tree Reconstruction

Our principal contributions in this case are as follows:

a) We designed a processing element that is efficiently able to compute the floating-point arithmetic operations and elementary functions related to ML function kernels.

b) We designed a NoC-based platform with a folded-torus network where each node contains a subnet of four crossbar-connected processing elements.

c) We designed novel job allocation schemes based on Hilbert space-filling curves to allocate nodes of the NoC to different requesting functions in a time-efficient manner with a view to minimizing overall application latency.

d) We explored and evaluated the performance of 3D NoC architectures, and demonstrated their superiority over 2D NoCs in terms of speedup and energy-efficiency.

e) We achieved function-level speedups of up to 847x, and aggregate speedup of the targeted kernels exceeding 6500x over baseline software runs. These represent more than an order of magnitude improvement with respect to existing hardware accelerator solutions.

1.1.2.1 Significance

Phylogenetic inference based on ML present a more challenging problem as far as hardware acceleration is concerned. Not only are their computation trees larger, each of their computation kernels involves a much larger amount of computation than in breakpoint phylogeny. In addition, all computations for these methods involve floating-point numbers, unlike those in MP phylogeny. ML methods have a wider usage among biologists and hence there has been considerably more research on this topic. These methods employ several computations of PLF through a class of functions we call phylogenetic kernels.

These kernels are composed of several instances of logarithm, antilogarithm and sum-of-products (vector product) computations. In terms of software run-time, the kernels account for more than 85% of the total run-time of the application. Our approach on this has been to distribute the kernel computations across cores of the NoC. This distribution is a deciding factor in determining the overall latency of the applications. Hence, novel methods of core allocation have been used and difference NoC integration approaches have been studied to achieve optimum performance.

1.1.3 High-Throughput, Energy-Efficient NoCs with On-Chip Wireless Links

Our principal contributions here are as follows:

a) We designed and evaluated NoC-based platforms for throughput-oriented scientific applications that consist of concurrently executing jobs with variable computation footprint.
b) We introduced long-range shortcuts in the NoC via wireless links and thereby reduced the average network diameter. We designed and evaluated several job allocation schemes based on this architecture.
c) We achieved a computation throughput of over 10^{11} operations per second, while consuming ~0.5 nJ for each such operation, thereby demonstrating that high throughput was attained without compromising on energy-efficiency.
d) We evaluated different architectures in terms of their power and energy consumption profiles, and established that the architectures delivering the highest throughput had favorable power and energy consumption profiles.

We also analyzed the correlation of throughput and power consumption with the statistical properties of the application traffic.

e) We carried out chip-level thermal profiling to identify hot-spot distribution and correlated them with architecture-level design tradeoffs.

1.1.3.1 Significance

High-performance scientific computing tools in emerging application domains such as biocomputing demand computation throughputs to scale to terascale and beyond. Given the diversity of tools and the need to cater to a wide user-base, it has become common practice, even within academic settings, to have a dedicated center which hosts a whole range of scientific computing tools on a few high-end data servers. The servers can be expected to service requests from a variety of applications, each with differing resource requirements, and simultaneously support them while delivering high throughput. This server could either be based on a cluster of general purpose microprocessors or make use of a co-processor consisting of a many-core chip where the cores are designed to accelerate targeted operations and are interconnected with an on-chip network.

The choice of the on-chip network architecture is an important consideration in the design of a NoC-driven platform targeted at enhancing computation throughput. Introduction of long-range links in regular architectures like mesh reduces the overall network diameter and improves inter-core communication latency. The use of on-chip wireless links to implement these shortcuts leads to significant savings in latency and energy, even

considering the overhead of wireless transceivers. In this perspective, our contribution here is the design and evaluation of NoC-based platforms with long-range on-chip wireless shortcuts to enhance the computation throughput of scientific applications.

1.2 Organization of the Dissertation

The remainder of the dissertation is organized as follows. Chapter 2 details the prior work in this field. It refers to existing work done for accelerating phylogenetic applications using one of MP, ML or BI methods. Chapter 3 treats the problem of MP phylogenetic tree reconstruction and the NoC-based solution. Sub-sections in chapter 3 deal with the problem statement, the algorithm used, design of the PE, network topologies, communication paradigm, application mapping and finally experimental results. Chapter 4 details the NoC-based platform design targeting ML phylogenetic tree reconstruction. It begins by providing a theoretical background of ML and mentioning the available software suites. This chapter subsequently describes our core architecture, NoC node, network topology, dynamic node allocation methods (including 3D NoCs), routing and arbitration, and finally experimental results. In Chapter 5, we propose a NoC-driven use-case model for throughput-oriented scientific applications, and subsequently use the model to study the effect of using long-range on-chip wireless links in conjunction with different resource allocation strategies on reducing the overall on-chip communication and enhancing computational throughput. In the experimental results, we compare these methods with respect to computation throughput, wireless link usage, energy and power consumption,

and chip-level thermal profiles. Chapter 6 describes areas that further research can explore. Chapter 7 is a list of references.

2. Related Work

2.1 State of the Art in Networks-on-Chip

Network-on-chip (NoC) is a paradigm that has recently emerged as an alternative to conventional bus-based point-to-point communication architectures, in order to deal with the increasing number of components in systems-on-chip (SoCs), higher demands on area and performance, and the limitations of global interconnects (high latency and energy consumption) with technology scaling. One of the earliest proposals for this paradigm can be found in Dally and Towles' seminal paper [23]. Ever since, there has been an ever-expanding body of work on NoCs in both academia and industry. An example of a NoC-based processor designed and manufactured at Intel Corp. can be found in [24].

The different aspects of NoC design that has received attention from researchers can be categorized into application-level, network-level and implementation-level. The reader can refer to [25], [26] and [27] to get a broad-based understanding of the issues involved.

At the application-level, traffic modeling has ranged from understanding the effect of synthetic traffic patterns [28] to modeling traffic generated by a popular application as self-similar traffic [29]. Parameter extraction from statistical traffic modeling has been described in [30]. A system of benchmarks for NoCs, which covers a wide spectrum of NoC design aspects, from application modeling to performance evaluation, has been presented in [31]. A statistical

approach to traffic modeling using traffic-load distribution plots that prevents overprovisioning for network link capacities is presented in [32]. A statistical physics-inspired approach to capture the capture the non-stationary traffic dynamics in multicore systems is presented in [103]. Closely related to this is the problem of application mapping on the NoC, or partitioning the NoC for multiple application requirements. Energy-aware mapping strategies have been considered in [33] and [34]. Incorporating floorplan information during application mapping is the subject of [35]. A methodology for mapping multiple use-cases on the NoC has been developed in [36]. There has been more recent work on partitioning the network for reducing message-level contention [37], bulk-synchronous parallel programming models [38], and virtualization and resource partitioning for traffic isolation [39]. Scheduling of applications on the NoC has been carried out with the objectives of enhancing performance (e.g. [40], [41]) and lowering power consumption (e.g. [42]), and have employed communication-aware voltage selection techniques (e.g. [43]). Similar work has been carried out for thermal optimization in 3-D NoCs [44].

At the network-level, a wide range of topologies have been proposed and studied. These range from simple topologies, such as ring [45] and 2-D mesh [46], to custom topologies built using heuristics [47], and complex topologies, including hierarchical star [48], mesh-of-tress [49], concentrated mesh [50] and other high-radix networks [51]. 3-D NoC topologies have been proposed and extensively evaluated in [74], [76], [77] and [78]. Introduction of long-range links in the on-chip network creates topologies that are neither regular nor completely random, essentially giving them a Small-World Property [116]. Different routing

strategies are explained in detail in [28] and [105]. They include both static and adaptive routing algorithms. Among static routing techniques, dimension order routing for mesh and e-cube routing for torus [52] are popular owing to their deadlock-free nature. Examples of other routing schemes are deflection routing [53] and oblivious routing [54]. For NoCs with long-range links, a deadlock-free routing strategy is south-last routing which limits the turns a routing path can take [115].

At the implementation-level, some of the important focus areas have been chip layout and metal routing [55] and integration of floorplanning and application mapping [47]. An EDA tool for 3-D NoC synthesis was proposed in [56]. Clock distribution has been one of the key focus areas, primarily deal with the problem of transporting clock signals across significant lengths of interconnects to all regions of the chip. An asynchronous clocking approach using encoded channels has been demonstrated in [57]. Mesochronous or globally asynchronous locally synchronous (GALS) clocking has been demonstrated in [24] and [58]. Recent work has focused on resonant clocking [59] and thermal-aware clocking [60] approaches. Another area that has seen significant contributions is power optimization. Various approaches, such as dynamic voltage scaling [61], on-off networks [62] and voltage-islands [63], have been proposed. A wide spectrum of power management methodologies has been reviewed in [64]. Reduced power budgets and increased sources of crosstalk bring into question the reliability of the on-chip communication network [65]. Some examples of more recent work focus on maintaining performance QoS by using

hop-by-hop retransmission checks and saving power [66] and graceful performance degradation in the presence of multiple link failures [67].

2.2 Hardware Acceleration for Phylogenetics

Substantial work has been carried out in the field of hardware acceleration targeted towards phylogenetics applications. These accelerators have been designed using platforms like FPGA, GPU, CBE and general-purpose multi-cores (traditional Intel/AMD dual-core, quad-core platforms). Most of the work targets probability-based methods like ML or BI because of their obvious importance to the biological community. There is some work on MP phylogeny, which is often serves as a quick reconstruction method to generate a lot of initial trees for "bootstrapping".

Mak and Lam [68] proposed a hybrid hardware/software system for solving the phylogenetic tree reconstruction using the Genetic Algorithm for Maximum Likelihood (GAML) approach. The genetic algorithm is implemented in software and the computationally intensive ML equation is implemented in hardware. This work uses a Xilinx Virtex XCV800 FPGA as the hardware accelerator and a Pentium 4 PC with 1 GB RAM for running the software. The likelihood function is evaluated in parallel in the dedicated FPGA. Their results while reconstructing a 4-taxa phylogenetic tree under the Jukes-Cantor Model demonstrate an overall speedup of 30 over software and an ML speedup of over 300, despite the communication overhead of the hybrid system. This work however does not explicitly state how the acceleration scales for larger taxa or more realistic complex models like GTR.

Alachiotis et al. explored the use of FPGA for accelerating the computation of PLF in [69]. A Xilinx Virtex 5 SX240T with 1056 DSP48E slices has been used. The DSP slices have been used to implement double-precision floating point multipliers and adders. Due to the limited amount of DSP48E slices on the FPGA, several multiplexer units are deployed to optimally exploit the available computational resources. A Sun x4600 system equipped with 8 dual-core AMD Opteron processors running at 2.6 GHz with 64 GB of main memory was used as the baseline. An average speedup of 8.3 over a single core has been demonstrated for trees comprising of 4 to 512 sequences on FPGA. The FPGA implementation also outperforms OpenMP-based parallel implementation on 16 cores in most cases, achieving speedups from 0.96 to 7.46. The projected computational time for a full tree traversal using Felsenstein's pruning algorithm for 512 taxa is less than 1 ms, based on reported clock speed of 284.152 MHz.

Bakos and Elenis [14] proposed a co-processor design for whole-genome phylogenetic tree reconstruction using a parallelized version of breakpoint median computation, which is an expensive component of a specialized form of MP phylogenetic tree inference (so-called breakpoint phylogeny). The co-processor uses an FPGA-based multi-core implementation of the combinatorial search portion of the TSP algorithm while the TSP graph construction is performed in software. The search tree partitioning is carried out in such a manner that each core explores the tree in a different order. This is done to avoid complex load-balancing and inter-core communication issues that occur if disjoint subtrees are assigned to different cores, because any of them might be subject to pruning. Their test system consists of 3.06-GHz Intel Pentium Xeon

processor and a single XilinxVirtex-2 Pro 100 FPGA connected to the host using a PCI Express interconnect. The best average speedup of 1,005 over software is observed using 3 cores. The best overall reduction in execution time is by a factor of 417. All these observations are for synthetic data and hence difficult to correlate with real-life examples.

Randomized Axelerated Maximum Likelihood version VI for High Performance Computing (RAxML-VI-HPC) [70] is an efficient parallel algorithm based on ML for phylogenetic tree inference. Blagojevic et al. have explored the porting, optimization and evaluation of RAxML-VI-HPC on CBE [71]. They carry out a detailed empirical optimization of RAxML on CBE, with additional support from the runtime environment. Different layers of parallelism have been used – task-level parallelism across SPEs, task vectorization within SPEs and/or loop-level parallelization across SPEs. It is shown that CBE outperforms both Intel Xeon and IBM Power5 and is more cost-effective and power-efficient than either architecture. However, the sheer complexity of porting the algorithm and the various optimizations required for CBE collectively pose a significant roadblock.

FPGA-based acceleration up to a factor of 10x has been demonstrated over software for Bayesian inference with MrBayes 3 tool in [72]. This paper describes a technique for mapping the PLF and supporting logic onto an FPGA-based co-processor. By leveraging the FPGA's on-chip DSP modules and the high-bandwidth local memory attached to the FPGA, the resultant co-processor can accelerate probability-based methods. The implementation achieves its performance by deeply pipelining the likelihood computations, performing

multiple floating point operations in parallel and through a natural logarithm approximation that is chosen specifically to leverage a deeply pipelined custom architecture.

In [73], MrBayes has been used on three different architectures to evaluate performance, scalability and programmability. General purpose multi-core (dual-core and quad-core Intel and AMD) processors and CBE support the Multiple Program Multiple Data (MPMD) model while GPUs support Single Program Multiple Data (SPMD) model. The PLF in MrBayes is parallelized using OpenMP directives for the general-purpose multiprocessors, POSIX threads for the CBE systems and Compute Unified Device Architecture (CUDA) for the GPU systems. For hardware-managed caches, the sharing of a cache level within the chip by all cores is a determining factor for efficient synchronization

Table 1: Performance comparison of hardware accelerators for phylogenetic inference

Phylogenetic tree reconstruction strategies	FPGA		GPU		Cell Broadband Engine		General Purpose Multi-core	
	Application speedup	*Total speedup*	*Application speedup*	*Total speedup*	*Application speedup*	*Total speedup*	*Application speedup*	*Total speedup*
Maximum parsimony (MP)	1005	417	-	-	-	-	-	-
Maximum likelihood (ML)	381	32	8.5	1.9	12	1.5	12	10

and hence scalability. Systems with software-managed caches like CBE compensate the user effort by efficient synchronization mechanisms. On the other hand, there are fewer data transfers between the device memory and CPU because GPU has sufficient memory to handle input data. CUDA automatically handles data transfer synchronization, thus relieving the user of the responsibility of providing any explicit synchronization mechanism. PLF

computation speedup is penalized by computation intensity and communication overhead inside the multi-cores. Quad-core AMD Opteron, where four cores are on a single die and share the same L2 cache, scales better compared to quad-core Intel Xeon, which has two L2 caches each shared by a pair of cores. For CBE, speedup values are close to ideal for small data sets and performance is stable across different computation intensities. Even though SPEs do not share a common cache, CBE is more tolerant to synchronization, primarily because it relies on user-generated software for this. However, speedup values for large data sets and computation intensities are almost equal for general-purpose multi-cores and CBEs. GPUs display an increase in speedup as the computation intensity increases because they are designed to perform efficient execution of small parallel threads in a scenario where the computation-to-data ratio is high. In terms of total frequency-normalized execution times, the general-purpose multi-core still achieves the best performance. This is based on the sum of the time spent in executing the parallel portion of the code (PLF) and that for the rest of the code. The degradation in total execution time for CBE is due to the fact that the PPE that handles the serial portion of the code is a rather simple core with a small cache, in-order execution capability and is burdened with the additional responsibility of synchronizing among SPEs. Table 1 summarizes the speedups achieved by different hardware accelerators for the MP/ML (application speedup) computation and the overall algorithm (total speedup).

Currently there are no NoC-based platforms that target phylogenetic tree reconstruction, whether MP, ML or BI. In this work, we propose and design hardware accelerators targeting MP (in particular, breakpoint) phylogeny and

ML phylogeny. In each of these cases, we show that our design has a superior performance to the state of the art in terms of absolute speedup provided.

3. NoC-Based Accelerator for Breakpoint Phylogeny

Maximum Parsimony is one of the two principal methods of phylogenetic tree reconstruction that we target in this work. The advantage of this method is that it provides a quick estimate to a phylogenetic tree structure. When the relative ordering of genes on a genome is known, a specific type of MP phylogeny called breakpoint phylogeny can be computed, based on the *breakpoint distance*. Prior work of designing hardware accelerators targeting breakpoint phylogeny is described in Chapter 2, e.g. [14]. In the following, we describe our NoC-based solution that delivers three orders of speedup over multithreaded software and one order of magnitude speedup over other hardware accelerators. To the best of our knowledge, this is the first comprehensive NoC-based solution for MP (breakpoint) phylogeny reconstruction.

3.1 Breakpoint Median Problem

Given a reference set of m genes $\{g_1, g_2, \ldots, g_m\}$, any genome can be represented by an ordering of the subset of genes that constitute it, as they appear from end to end of the genomic DNA. The *breakpoint distance* between any two genomes is defined as the number of gene pairs that appear adjacent in one genome but not in the other. It is a measure of how different two genomes are by their gene ordering. For example, let us consider two hypothetical genomes $G_1 = g_1g_2g_3g_4g_5$ and $G_2 = g_2g_3g_5g_4g_1$. According to the definition above, the number of breakpoints between *G1* and *G2* is 2, if we do not consider circular adjacency. We can find a set of breakpoint distances where each distance pertains to a pair of genomes in the input set. Blanchette et al. pioneered the

work on breakpoint-based phylogeny [3]. They reduced the problem of constructing an optimal phylogenetic tree of N genomes to one of solving numerous instances of a version of the Traveling Salesman Problem (TSP) [4] where edge-weights of the input graph are bounded to a fixed set of integer values. Put intuitively, each instance of TSP tries to identify the gene order of a hypothetical ancestral genome that is the closest representative to any three given genomes. This problem is called the *3-median breakpoint problem* and has been proven to be NP-Hard [5].

3.2 Algorithm

TSP is a well-researched problem and various approaches have been proposed in literature to solve it. These algorithms can be classified into two groups – (a) approximation algorithms that could take polynomial time [15], [16], [17], [18] and (b) accurate algorithms that run in super-exponential time [21], [22]. Techniques used in approximation methods include the Kernighan-Lin heuristics, simulated annealing and genetic algorithms [15], [16], [17], [18], [19]. Among accurate methods, dynamic programming [22] is super-exponential in practice, whereas *branch-and-bound* methods [21], [22] achieve significant pruning of search space during computation without affecting the optimality of the output. This method, actually a run-time heuristic, is computationally intensive but is easily parallelized. Coarse-level parallelization of TSP has been explored using genetic algorithms [19] and branch-and-bound [1], [20].

3.2.1 Branch-and-Bound Method

In this section, we present the core computation steps of the branch-and-bound run-time heuristic to solve TSP that we used in our implementation. The input is a directed graph, $G = (V,E)$ with m vertices and a non-negative cost associated with each edge. The m vertices of this graph correspond to the m reference genes and its edges have a bounded weight – an integer cost between 0 and 3, or an edge with cost ∞ (representing nonexistent edges) [3]. The output is a least cost cyclic tour that traverses all vertices exactly once.

The overall algorithm has a worst-case runtime complexity that is super-

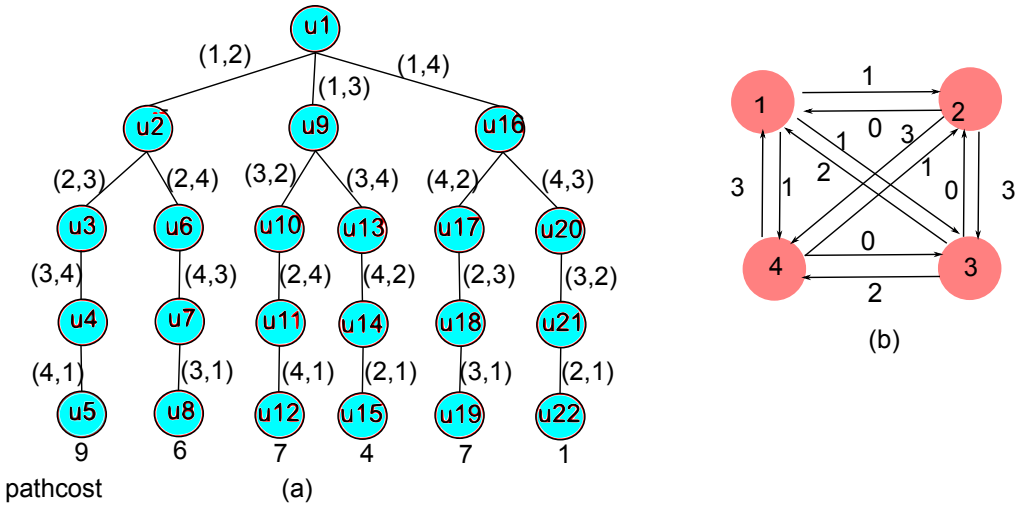


Figure 3-1: An example showing (a) the exhaustive search tree corresponding to the input graph in (b). If the tree is computed in the Depth First Search order, then evaluation of the path that leads to a low cost (such as u1-u2-u6-u7-u8) first may help in pruning the computation of a higher cost path (such as u1-u9-u13-u14-u15). This idea is exploited in the branch-and-bound technique.

exponential in the number of vertices (i.e., genes). However, the use of branch-and-bound technique reduces this search space significantly for most practical inputs. For example, for m=110, which represents a typical input genome size for

bacterial genomes, the theoretical number of "reductions" is ~10^{178} while the maximum number of such operations observed in our experiments in ~10^{9}.

Given this input graph *G*, the solution space can be represented by a conceptual computation tree. An example is shown in Fig. 3.1. The tree has a total of *(m-1)!* potential paths to be explored before identifying the optimal TSP tour. Every tree-edge *(u,v)* from a parent node *u* to a child node *v* corresponds to a graph edge *(i,j)* ∈ *E*, and every path from the root to a leaf node encodes a completed TSP tour with cost equal to the sum of the edge weights along its path. An optimal TSP tour represents a least-cost path. Our algorithm dynamically generates and explores this conceptual search-space tree in the

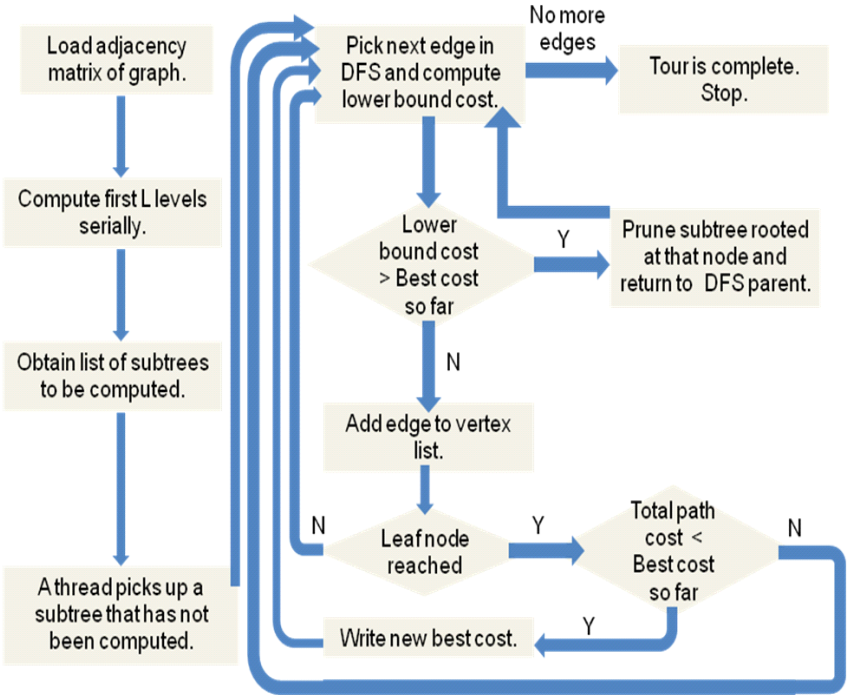


Figure 3-2: Flow diagram showing steps of the branch-and-bound method

depth-first-search (DFS) order.

Initially, a global variable called *best_cost* is initialized to ∞; this variable is dynamically updated to keep track of the least cost over all TSP tours

examined so far at any stage of the algorithm. At every step, the algorithm evaluates the next eligible tree-edge in the DFS order as explained below and also shown in Fig. 3.2.

At any given step, consider the newly included tree-edge to be from node u to node v, and the cost of the corresponding graph edge (i,j) to be c_{ij}. Let $c^*(v)$ denote the cost of the least cost TSP tour passing through node v. There are two possibilities for v:

If v is a leaf, then $c^*(v)$ is set equal to the net cost of the path from the root node to v. Subsequently, if $c^*(v) < best\_cost$ then $best\_cost$ is updated to $c^*(v)$.

If v is an internal node in the search tree, a lower bound for $c^*(v)$ is computed using a matrix reduction operation. If the computed lower bound ($lbc(v)$) is observed to be greater than or equal to $best\_cost$, further exploration of the subtree under v becomes unnecessary and so the subtree is pruned and the computation returns to the parent node u; otherwise, the DFS is continued under v's subtree.

3.2.1.1 Lower Bound Calculation

We use the method shown in [22] for lower bound computation at each tree-edge. An $m \times m$ matrix called the *reduction matrix (R)* is maintained throughout execution. Initially, the matrix at the root node is set equal to the cost matrix defined by E. At any step of the DFS, $lbc(v)$ is calculated as follows:

1) All entries in row i and column j of R is set to ∞;

2) $R[j,1]$ is also set to ∞;

3) All rows and columns that contain at least one non-infinity value are *reduced* as follows:

(a) Given row i, compute $min_i = min\{R[i,j]\}$ for all $1 \le j \le m$;

(b) Then for all $1 \le j \le m$, $R[i,j] = R[i,j] - min_i$;

(c) Similarly, given column j, compute $min_j = min\{R[i,j]\}$ for all $1 \le i \le m$;

(d) Then for all $1 \le i \le m$, $R[i,j] = R[i,j] - min_j$ As this is done, all subtracted values (i.e., the minimum values) are accumulated into another variable *adjCost*.

4) Subsequently, the lower bound is given by: $lbc(v) = lbc(u) + R[i,j] + adjCost$.

3.2.2 GRAPPA

A software suite called Genome Rearrangements Analysis under Parsimony and other Phylogenetic Algorithms (GRAPPA) [85] computes an exhaustive search across all possible trees for k taxa ($3*5*7*\ldots*(2k-5)$ trees) and iteratively runs multiple instances of a TSP solver for scoring each tree. It is widely popular for MP phylogenetic tree reconstruction in software and is used as the basis for hardware acceleration in [14]. GRAPPA can be run in single-threaded and multi-threaded modes. We use multi-threaded GRAPPA runs as reference for benchmarking the performance of our NoC-based solution.

3.3 Core Architecture: PE Design

The problem of MP (breakpoint) phylogenetic tree reconstruction using branch-and-bound technique naturally lends itself to parallelization using a divide-and-conquer approach by subdividing the solution-space tree into

independent subtrees. A PE computes one subtree at a time and considers pruning based on the best cost available from its peers. As this requires a good combination of parallelism and inter-core communication, NoC provides an ideal platform owing to its inherent parallel architecture, customizability of its core and its efficient communication infrastructure. We designed and implemented the PEs and the on-chip communication network for this NoC. Two types of communication infrastructure were explored. One is a regular mesh network. The other is a hierarchical four-way tree or quad-tree.

The PE has a pipelined architecture optimized to handle the computation along an edge as per the algorithm described in 3.2.1. Since the PE carries out the most computationally intensive part of the whole operation, our attempt has been to optimize its architecture to ensure that the number of clock cycles required scales nicely with increasing graph size (number of vertices, m). The primary performance parameter is timing, which aims at reducing application run-time and overall latency. To achieve this, we designed our PE for $O(m)$ time complexity, as discussed further in 3.3.1. Our PE has an integer datapath because breakpoint median computation for MP (breakpoint) phylogenetic tree reconstruction consists entirely of integer operations. The principal components of the PE are a *reduce* block and peripheral control logic, each of which is described in detail below. We use the short-form $lg\ k$ to denote $log_2 k$. The datapath consists of the following fields (m: number of vertices, w: maximum edge weight).

a. x – the parent node (u) uses $lg\ m$ bits

b. *y* – the child node (*v*) uses *lg m* bits

c. *LBC* – the lower bound cost (*lbc(u)*) estimate at an edge; this requires *lg m + lg w + 1* bits

d. *EPC* – the exact path cost (*lbc(u)+R[i,j]*) determined so far; takes *lg m + lg w + 1* bits

e. *TSP* – the TSP adjacency matrix (*R*), flattened. Its representation takes $m^2 * lg\ w$ bits.

f. *VLST* – the current list of vertices traversed; *m*(lg m) + 1* bits are required to store this field.

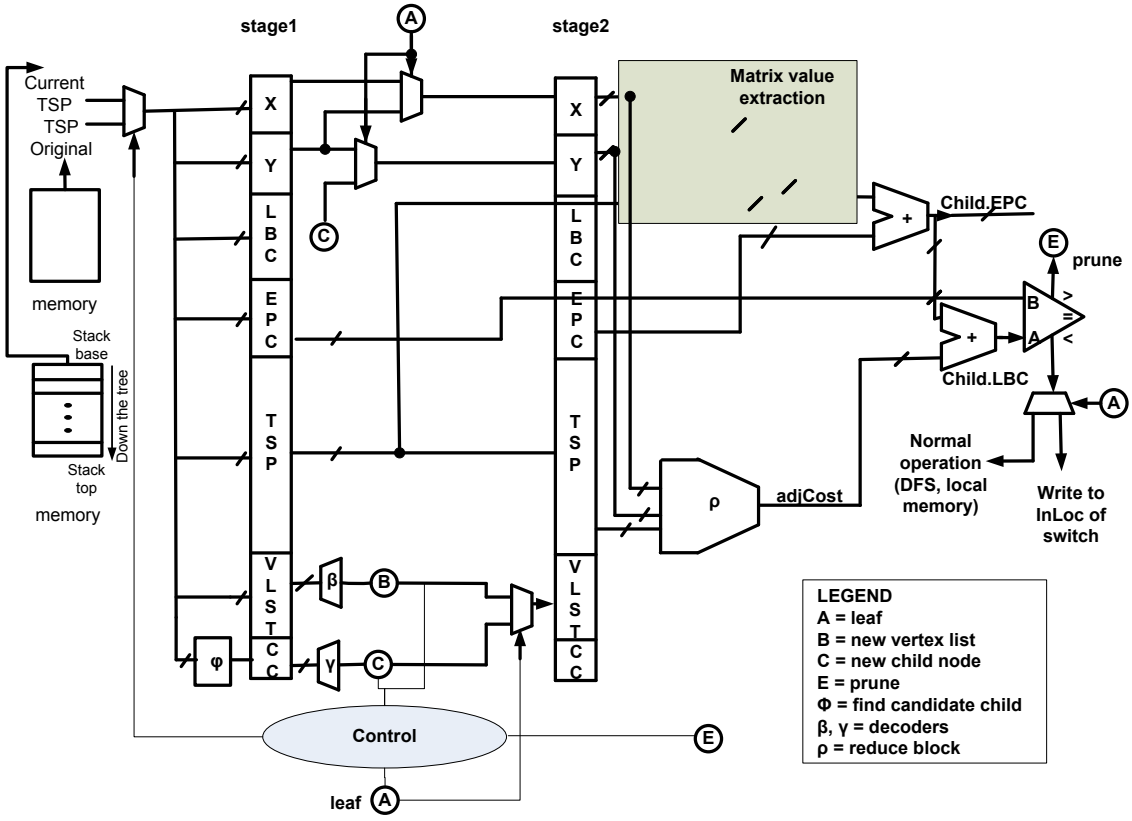


Figure 3-3: Internal architecture of processing element

g. *CC* – the candidate children at every stage; takes *m* bits

As is evident, the datapath complexity of the hardware is $O(m^2)$. In our approach, breakpoint distances can range from 0 to 3, which is the range of the valid weights we used. We used the weight 4 to denote a non-existent edge or ∞. A different range of weights just changes the number of bits for *w*. A block diagram of the PE is shown in Fig. 3.3. Subsequent references to the sub-blocks in parentheses (e.g. ρ, φ, etc.) in this sub-section refer to this figure.

3.3.1 Reduction Block

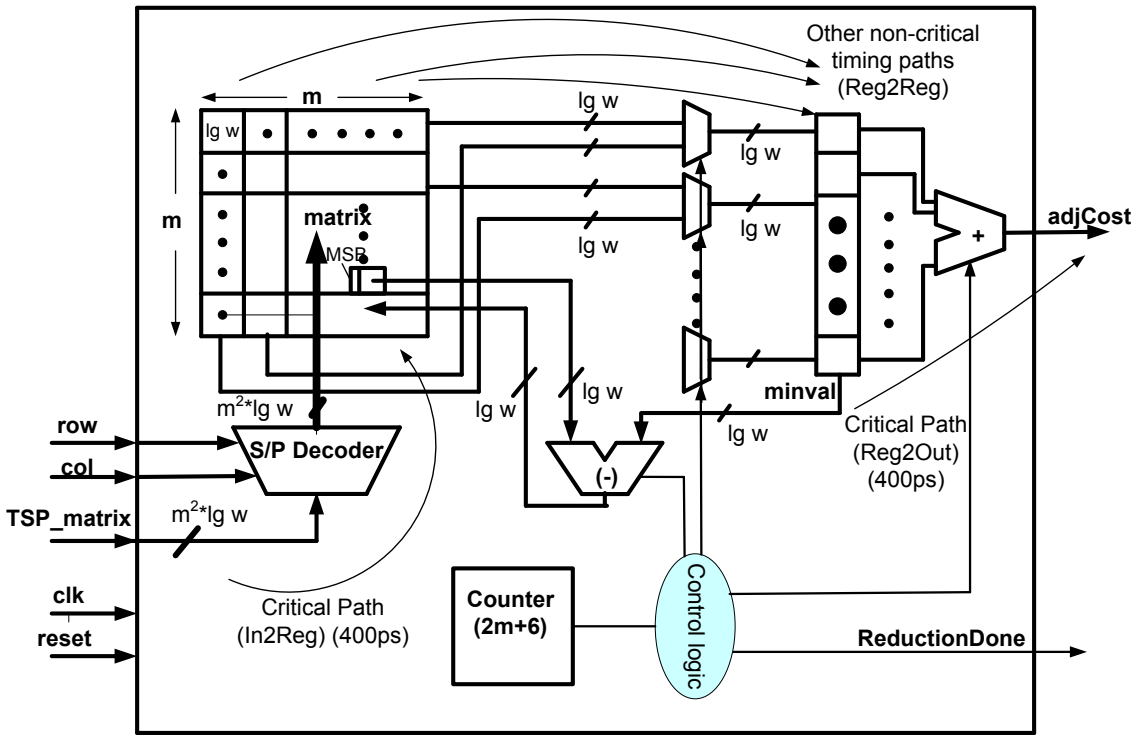


Figure 3-4: Internal architecture of reduce block (ρ) for linear-time matrix reduction

This block (ρ) carries out the matrix reduction operation described in 3.2.1. Based on the algorithm, the run-time of the operation is a function of the matrix size, i.e., $O(m^2)$. This operation consumes the maximum fraction of the total time

required for an edge computation. Hence, a significant amount of time is saved by suitably optimizing its design. Our implementation achieves *O(m)* cycle time by using micro-level parallelism inside the *reduce* block. This has the effect of drastically reducing the total time as well as providing better time-scalability with increasing input graph size, *m*.

The matrix is reduced using the new values of *x* and *y* in *stage2* (see 3.3.2 for details on the operations up to this stage) and the adjacency cost *adjCost* is obtained. Fig. 3.4 shows the architecture of *reduce* block. The flattened TSP matrix is initially reorganized into rows and columns in the component denoted as *matrix*. There are *m* rows and *m* columns with each entry taking up *lg w* bits. The register bank *minval* of width *m*(lg w)* is initialized with a bit pattern representing *infinity* (3'b100 as mentioned earlier). A *counter* is used as a state machine controller. There is an *m*-sized bank of comparators that compare one element from every row or column in every cycle. Minimum value calculation for all rows and the same for all columns take *m* cycles each. Additional three cycles are required for subtraction of the minimum values, for calculation of the final *adjCost* and for control operations for each case (row and column blocks). The entire reduction operation takes *2*(m+3)* cycles to complete.

3.3.2 Peripheral Control Logic

The peripheral control logic is used for vertex selection, cost comparison and data management. The register bank for the first stage is *stage1*, which has the same width as the datapath. The input control multiplexer initially switches

to select the current vertex data. The *CC* field is computed (φ) from *VLST* in *m* cycles in the worst case.

In the second stage, the candidate child is found by scanning (γ) *CC* of *stage1*. Again, this requires *m* clock cycles in the worst case. Using this candidate child, *VLST* is updated (B) for the child node in the graph. If it is not a leaf node (A), the candidate child becomes the next child node, while the current node (*y* of *stage1*) becomes the parent node *x* of *stage2*. During the same stage, the data pertaining to the best case obtained so far is fetched into *stage1*. The input multiplexer now selects the lowest cost data (*global best cost*) available to the PE at this time. At this stage, *TSP* of *stage1* gets the original TSP matrix.

The current value of the exact cost of the path found so far, *EPC* is updated by adding to it the edge cost from *x* to *y* in the original adjacency matrix. This is checked against *global best cost* and *reduce* operation is started only if *EPC* is lower. The sum of *adjCost* (obtained from *reduce* operation) and *EPC* yields the lower bound cost, *LBC*, which is again compared with the best cost found so far. If *EPC* or *LBC* is larger than the current *best cost*, the tree is pruned (E), the current child is aborted and the path through another child is explored. The data on *stage2* is reloaded back to *stage1* with the old value of *x* and a new calculation for the candidate child. If *LBC* is smaller and we have not reached a leaf node, normal operation (DFS) continues with the new set of data. If we have hit a leaf node with an *LBC* lower than the best cost globally found so far, this value (new *global best cost*) is sent to the switch to be communicated with other PEs in the network.

3.3.3 Memory

The memory is physically distributed across all PEs, and the memory local to each PE has two logical partitions. One part of the memory stores the TSP matrix corresponding to the root of the subtree that is currently assigned to that PE. Another part of the memory stores the intermediate matrix data that result along the way of evaluating a path down that subtree.

Table 2: Per PE memory requirement for different input genome sizes

Number of genes per input genome, m	Per PE memory requirement (MB)	
	Basic scheme	Improved scheme
128	0.514	0.024
256	4.063	0.094
512	32.282	0.375
1024	257.252	1.5

The part of the memory that stores intermediate matrix data can be implemented as a stack. During DFS, the new vertex data (path cost, vertex list and associated adjacency matrix) are pushed into the stack (Fig. 3.3). The stack is full only when the leaf node is reached. If there is pruning (before the leaf node is reached), the stack is popped. In this scheme, every PE has a stack with m levels, where each level of the stack needs to store $(m+1)*(\lg m) + (m^2+1)*(\lg w) + 2$ bits. Since $\lg w$ is a constant, the total memory requirement is $O(m^2)*O(m)$ or $O(m^3)$. The total memory required per PE for different values of m are shown in Table 2.

An improved scheme is explored, where the memory requirement is reduced to $O(m^2)$. In this scheme, the adjacency matrix at each level is not stored in the stack. Instead, we store only the original values in the row i and column j

that are made ∞, the row-wise minima and the column-wise minima obtained during reduction at each level. These data require $4m*(\lg w) + 2m$ bits at each level. In addition, the adjacency matrix only for the current child level is stored. While going back to the parent, these data at each level are used to backtrack and reconstitute the adjacency matrix at the parent level. The reconstitution step leads to a negligible run-time penalty (1.8%) but the overall memory requirement improves to $(m+1)*(\lg m) + (5m^2+1)*(\lg w) + 2m^2 + 2$ bits. This improves the memory-scalability of the design and enables implementation for higher values of m for the same per-PE memory as can be seen from Table 2. We use this memory implementation for our experiments.

A list of all subtrees to be computed is maintained in memory. Once each PE completes one subtree reduction, it picks up the next available subtree and removes it from the list. This is achieved by maintaining a global array of flags and a mutually exclusive semaphore.

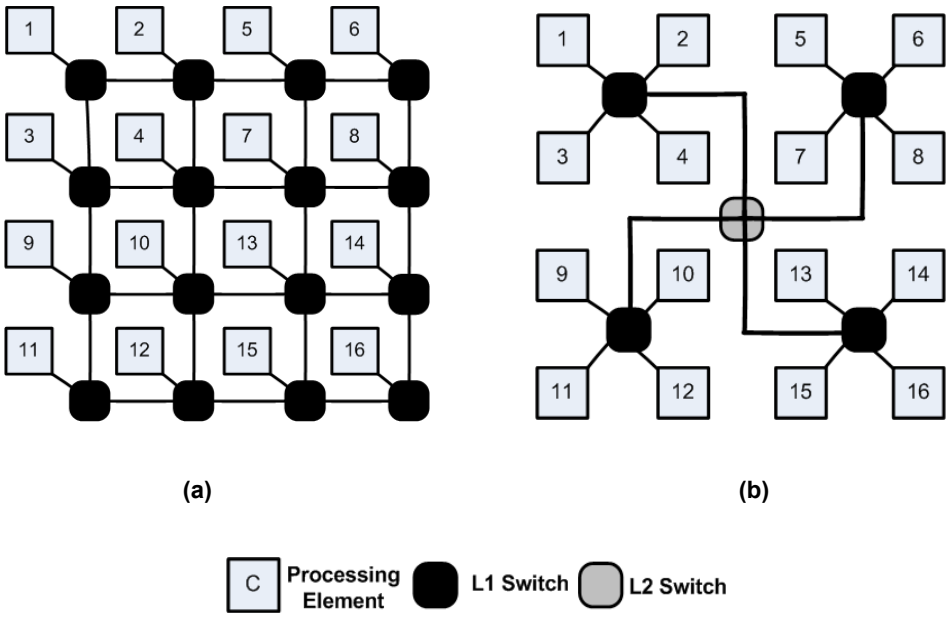


Figure 3-5: (a) Mesh network architecture (b) Quad-tree network architecture

3.4 Network Architecture

The choice of the network architecture is affected by application modeling and traffic pattern analysis, as is explained in the seminal paper on NoC design methodology [27]. Our application is mapped on a set of homogeneous cores, each of which carries out reduction of a subtree. The need for communication arises when a PE needs to update the network with the best score it has obtained. This is explained in detail in this section and the next. The mode of communication involved in this case is a *conditional broadcast*. We explored two different kinds of network architecture – a mesh, shown in Fig. 3.5(a) and a quad-tree, shown in Fig. 3.5(b). A mesh is the most appropriate scalable topology for broadcast traffic. Its regularity provides for easier timing closure and reduces dependence on interconnect scalability [27]. The hierarchical nature of a quad-tree minimizes the diameter of the network for the same number of nodes, thereby amortizing the router (switch) overhead and reducing latency [27].

Other common network architectures like point-to-point, full crossbar or ring do not scale well with increasing system size [86], [87] and far exceed latency/area budgets. With increasing system size *(N)*, the number of inter-switch links in a mesh increases faster than that in a quad-tree. The expected volume of inter-PE communication in our application is relatively low. Hence, having fewer links in our network can lead to potential savings in area and power without incurring a risk of network congestion.

The diameter of a mesh architecture increases as $O(\sqrt{N})$ where N is the system size or the number of PEs. The same for a quad-tree increases as

Table 3: Wost-case write latency in clock cycles

N	Mesh	Quad-tree
4	6	6
8	9	10
16	12	10
64	14	12
256	30	14
1024	62	16

$O(log_4 N)$. As mentioned earlier, the mode of communication for our application involves some form of broadcast as the *best cost* is written to all the PEs except for the originating PE. Hence, the worst-case hop count is a linear function of the diameter. It should be remembered that all links are not of the same length in a quad-tree, where links higher up the tree are longer and have greater delay. Table 3 shows an estimate of the number of clock cycles required per write in the worst case in 65 nm CMOS technology with a clock period of 400 ps. Quad-tree has an advantage over mesh in terms of communication latency for N>16. However, the key advantage of a quad-tree comes from power savings because the number of links and switches is drastically reduced. These observations are made on the basis of the experimental results reported in 3.7.

The problem of partitioning the application and mapping it to the nodes of the NoC is also important in optimizing overall latency. This is discussed in detail in 3.6.

3.4.1 Mesh Switch Design

A typical switch for the mesh network architecture is shown in Fig. 3.6 (a). Input buffers *InN*, *InE*, *InS*, *InW* receive data from four neighboring switches and input buffer *InLoc* receives data from the associated PE. There is a

dedicated buffer (*BufOut*) that provides data to the network as well as to the associated PE. Each set of input/output data consists of the fields (a) Path Cost, (b) Vertex List and (c) Transmission control bits. At every cycle, one of four transmission decisions are taken by the Decision Making Unit (DMU) and the data is written into an internal buffer (*local*). The same is transmitted out in the next cycle through *BufOut*. The transmission control bits are as follows.

NOTX: No valid transmission

NORETX: No retransmission

DOTX: New best cost from local PE; transmit

TRWL: New best cost from other PE; transmit and update local PE

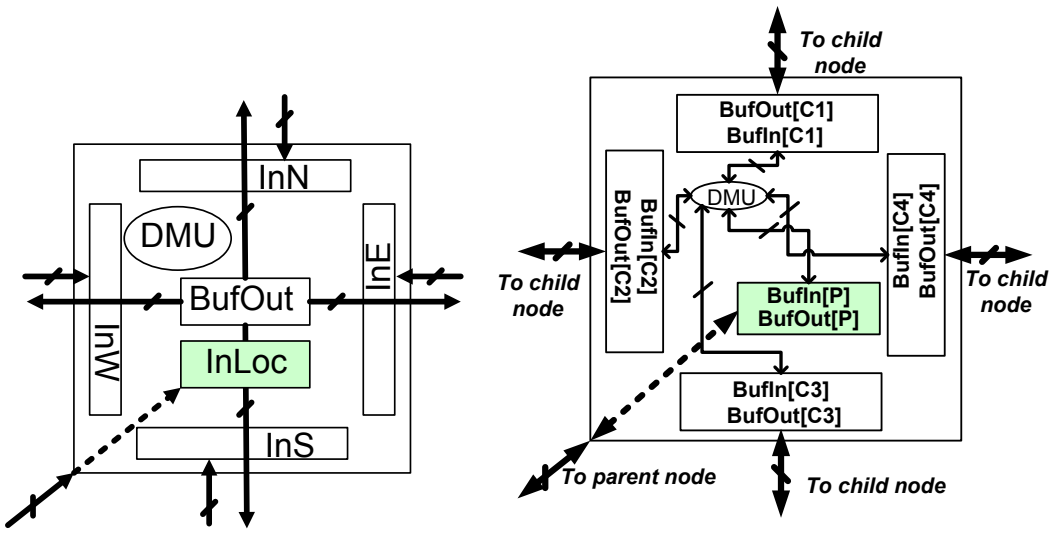


Figure 3-6: Internal architecture of (a) mesh switch and (b) quad-tree switch

Fig. 3.7 shows all the control states for decision making in a mesh switch and Fig. 3.8 shows the timing diagram for a typical situation. It is to be noted that a switch receives data from each of its neighboring switches in every cycle but the transmission control bits determine whether the data is valid for

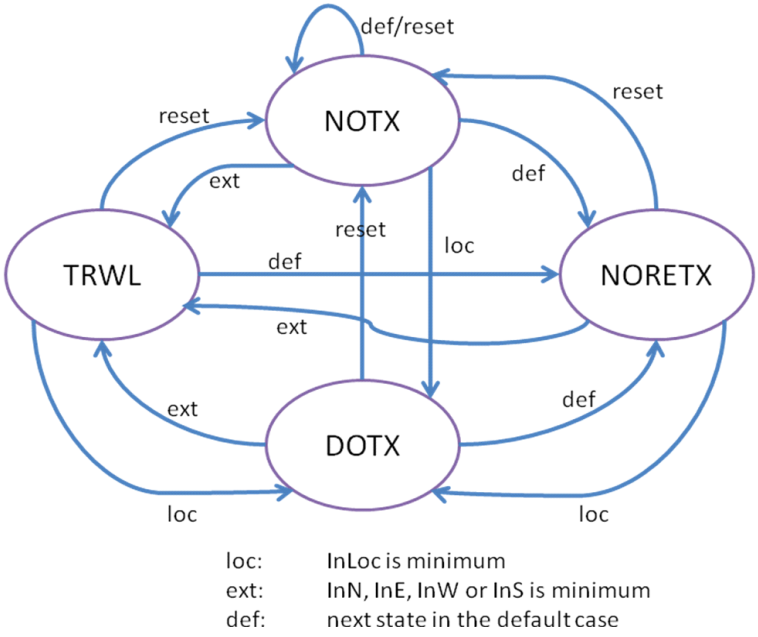


Figure 3-7: State diagram of control states in a mesh switch

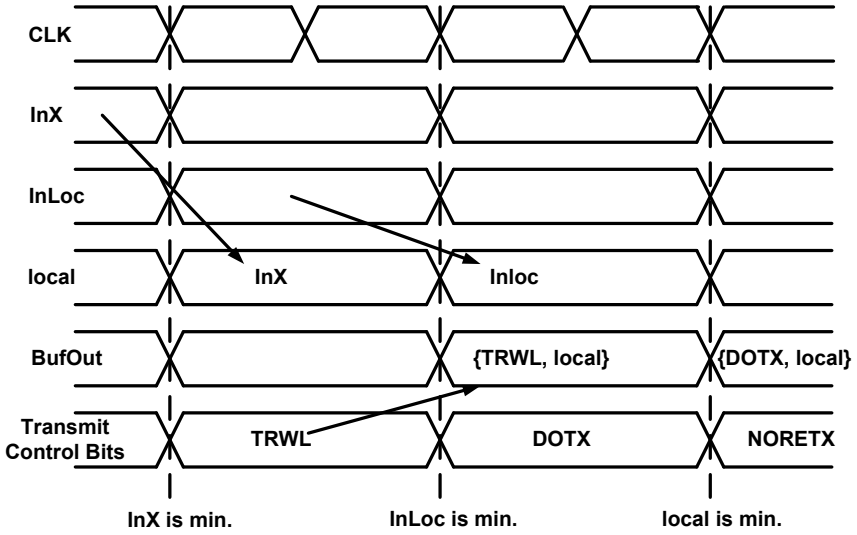


Figure 3-8: Timing diagram showing typical scenarios encountered in a mesh switch

consideration or not. The data is considered if the control bits are *DOTX* or *TRWL* but not if they are *NOTX* or *NORETX*.

3.4.2 Quad-tree Switch Design

There are different levels of switches for this network architecture. The leaf level switches (refer to Fig. 3.5(b)) are denoted L1, the next higher level L2 and so on. An L1 switch consists of five buffered input/output ports (*BufIn/BufOut*), four catering to the four leaf PEs and the fifth to the parent switch. For an L2 switch and upwards, four children ports cater to lower level switches and the parent port caters to the higher level switch. The top level switch has only four downlinks but no uplink. Each set of input/output data consists of the fields (a) Path Cost, (b) Vertex List and (c) Update control bit (*UCB*). The switch architecture is shown in Fig. 3.6(b). *UCB* is a flag to indicate whether the status of the data is valid (*UPDT*) or invalid (*NOUP*). The receiving parent or child switch infers "no transmission" if *UCB* is set to *NOUP*. In every cycle, the switch takes a decision based on the following algorithm.

Let *C1*, *C2*, *C3* and *C4* be the four (children) downlinks and *P* be the (parent) uplink and let us define the set $L = \{C1, C2, C3, C4, P\}$. Let us suppose the best (lowest) path cost, PC_i for a decision cycle comes from $i \in L$, i.e., $PC_i < PC_j \ \forall j \in L, j \neq i$. Then, we have the following set of assignments.

$BufOut[k] \leftarrow PC_k \ \forall k \in L$

$UCB[i] \leftarrow NOUP$

$UCB[j] \leftarrow UPDT \ \forall j \in L, j \neq i$

3.5 Communication Paradigm

In both network architectures, every PE communicates with its neighbors through its local switch. In the mesh architecture, every switch communicates with its immediate neighbor and gets data in every cycle from at most four neighboring switches. Based on the decision mechanism described in 3.4.1, the switch places data on *BufOut* with appropriate control bits. The neighboring switches get this value in their input buffers in the next cycle. Hence, at every cycle, data is sent in all four directions.

In the quad-tree, every switch communicates with its four children and one parent in every clock cycle. It receives data from its parent and/or one or more of its children and takes a decision on the lowest cost available to it thus far. Once found, this data is placed on four output buffers, except the direction it came from along with appropriate *UCB*, as described in 3.4.2. For the best-cost data to propagate to the entire network, it has to go through a maximum of H hops where H is given by

$$H = 2 * \lceil \log_4 N \rceil \qquad (1)$$

Note that *H/2* is the *height* of the tree. One important fact to keep in mind is that each hop does not consume the same number of clock cycles as the wire length varies at different levels.

The need for inter-PE communication arises when a particular PE checks against the *global best cost* obtained so far and finds out that its local best-cost is lower than the global best-cost. At this stage, the PE should broadcast its newly obtained value to the whole network. One way to implement this is to use

flooding. However, this could lead to an unnecessary network congestion thereby affecting scalability. Therefore, we devised an improved alternative strategy where a PE *conditionally broadcasts* valid data only if

a. Its local best-cost is worse than the global best-cost but it has *not yet participated* in the broadcast of this global cost, or

b. Its *local best-cost* is better than the *global best-cost* (currently available to the rest of the network) *and* it has *not been previously transmitted*.

The above scheme ensures elimination of redundant communication, thus reducing communication overhead and power consumption without compromising on the correctness of the answer.

3.6 Application Mapping and Tradeoff

Partitioning and mapping the application on the NoC has a significant

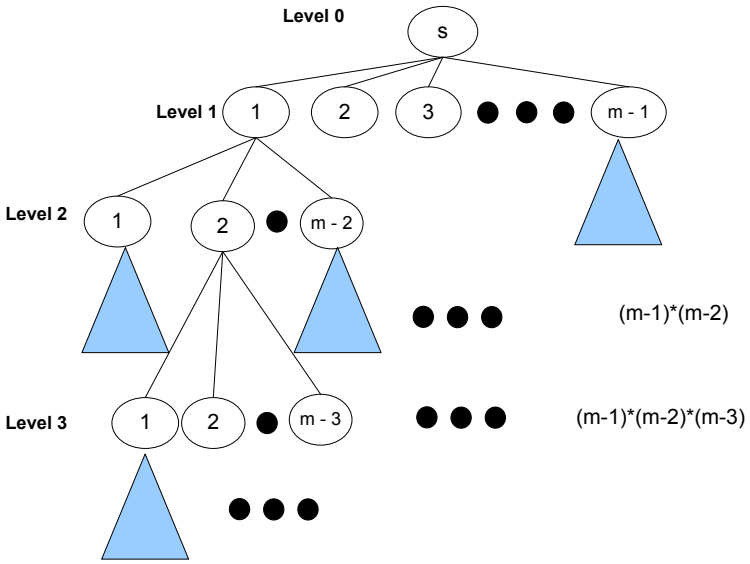


Figure 3-9: Number of subtrees generated by partitioning the search-space tree at different levels

impact on its overall latency and total energy consumption. The PE that finishes its share of reduction computations last limits the performance of the entire system. The determining factor for this is the load distribution among PEs, which is dependent on input data. In our scheme, each PE picks a subtree dynamically from a common pool of available uncomputed subtrees, once it has finished computing its own subtree. This can happen either when the PE has finished computing the subtree exhaustively or when it has pruned it. This could result in each subtree contributing to a different number of reductions and each PE computing a different number of subtrees. It is evident that we need to ensure that PEs are evenly utilized to minimize the impact of a "bottleneck" PE and achieve the best overall latency. Hence, application mapping on the NoC needs to be optimized such that the load distribution among PEs is even. We use the following definitions to formulate the problem.

f_i: utilization factor of PE i

t_i: application latency of PE i

T_{PCIe}: latency overhead of the system for loading data through PCIe

T_{pick}: cumulative latency overhead of each PE picking subtrees from the pool of uncomputed subtrees

Overall application latency, $T_{overall}$, is given by

$$T_{overall} = T_{PCIe} + T_{pick} + max\{t_i\} \qquad (2)$$

where $max\{t_i\}$ is over all i. Note that the last term is the latency of the "bottleneck" PE. t_i is proportional to f_i. T_{PCIe} and T_{pick} are related to f_i as explained below.

Experiments showed that the load distribution among PEs (f_i) becomes more balanced when the number of subtrees in the common pool is much higher than the number of PEs. For a graph with m vertices, the solution-space tree with the starting node as root (level 0) has $(m-1)$ nodes at level 1, $(m-1)*(m-2)$ nodes at level 2, $(m-1)*(m-2)*(m-3)$ nodes at level 3 and so on (Fig. 3.9). In general, it will have ${}^{m}P_{k+1}$ nodes at any level $k < m$. So partitioning the solution space by choosing subtrees rooted at a deeper level generates more subtrees, helping to balance load and thereby ensure maximum achievable parallel speedup. Now, T_{PCIe}, the overhead involved in loading the entire set of subtrees to the system using PCIe increases with the amount of data that needs to be transferred, which increases with the number of subtrees. T_{pick} also increases with the total number of subtrees handled by each PE. Hence it is clear that the dependence of t_i on f_i is opposite to that of T_{PCIe} and T_{pick} on f_i. We need to optimize $T_{overall}$ in (2) with these constraints. As explained in 3.7.1, we have considered m=110 in our experiments. In this case, we resolved this tradeoff by choosing to work on subtrees rooted at level 2, which generated 109*108 subtrees and yet kept the overhead to a manageable amount. Note that 109*108 (=11772) is much larger than the largest system size (number of PEs, N=64) we experimented with, which led to a balanced load distribution.

3.7 Experimental Results

3.7.1 Experimental Setup

The performance evaluation of the NoC was carried out from the timing and power perspectives during phylogenetic tree reconstruction with varying data sets. Different parameters associated with the NoC are as follows. The system size, N, is the number of PEs in the NoC. N was set to 4, 16 and 64 for evaluating the performance of the NoC with scaling of system size. The number of vertices in the input graph is denoted by m, which determines the width of the datapath. In practice, this value should be set to the number of genes shared by the input genomes. For example, chloroplast genomes of potato, tomato and wheat share 110 genes; hence m=110 in this case. In our experiments, we used two types of input data: (a) multiple sets of synthetic genomes with m=110 used for exhaustive system-wide parametric study; and (b) two sets of real input genomes (as explained in 3.7.3). Note that the value of m affects the size of the datapath and the memory requirements in the PE as per the discussion in 3.3. Since we have dealt with three-median breakpoints, breakpoint distance can vary between 0 and 3. Without loss of generality, the maximum weight w has been taken to be 4 to indicate ∞ or a non-existent edge. As with m, this choice affects the datapath size but to a lesser degree.

Each PE with its corresponding switch constitutes one node in the NoC. They were implemented by synthesizing Verilog RTL using Synopsys Design Compiler followed by place-and-route with Cadence SoC Encounter using standard cell library of 65 nm process [88]. Extracted parasitics were used in

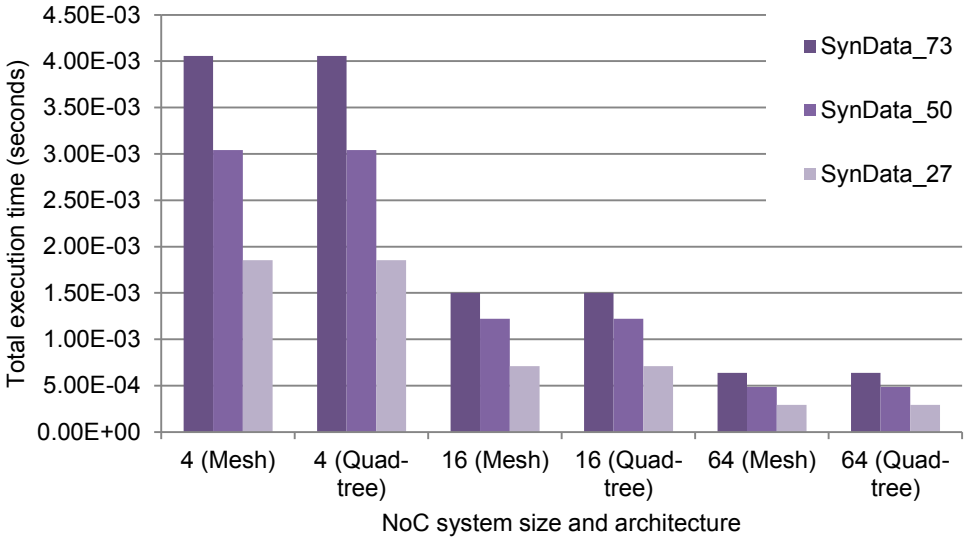
Synopsys PrimeTime to determine post-layout timing performance. The pipelined design could sustain a clock frequency of 2.5 GHz in each PE and switch. This was verified with m=110 and higher. The critical path delay using 65 nm timing library is within 400 ps, as shown in Fig. 3.4. In order to estimate the total power dissipation, it becomes necessary to record the total communication events involving all the PEs. For modeling the event statistics, we implemented a multithreaded program to act as the *software driver*, which recorded the number of reduction operations performed by each thread, and the number of successful write operations by that thread. Each individual thread of the software driver functionally simulated a processing element of the NoC. Thereafter, these statistics were used in conjunction with Synopsys Power Compiler using the library [88] for estimating the total computation power of all the PEs. The switch power (also obtained from Synopsys Power Compiler) was separately added to this component. Logic gate count for one PE and associated network switch with m=128 is 1.267 million.

Interconnect characteristics (delay, power) were determined using Cadence Spectre. Wire capacitance information extracted from layout was used to determine delay and energy dissipation of interconnects. Multiple clock cycle delay in longer interconnects was accounted for.

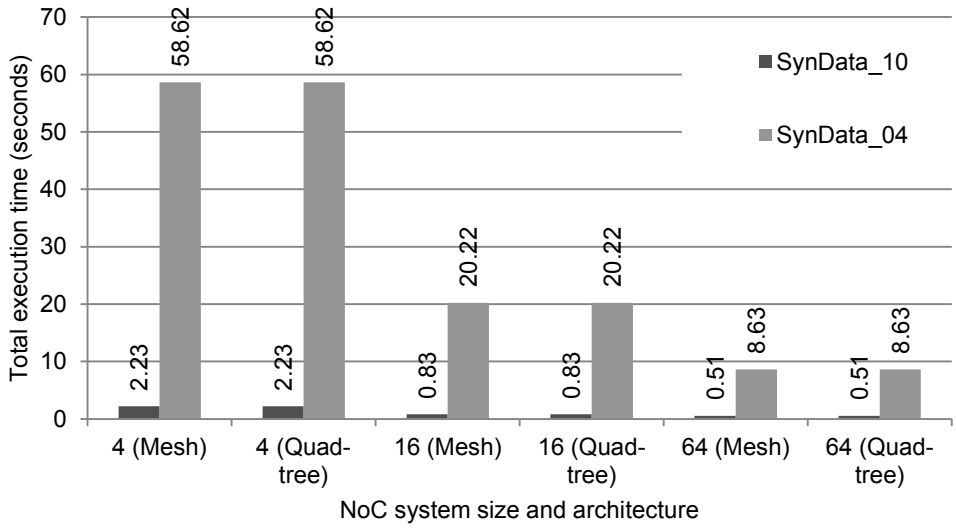
PCI Express 2.0 is used as the interface for initially loading the graph data into the NoC. For modeling this interface, Synopsys Designware IP PCI Express 2.0 PHY was used. It has been implemented on 65 nm process and operates at

5.0 Gbps. We use a 32-lane PCIe 2.0 for our simulation. Both mesh and quad-tree architectures were considered for performance evaluation.

GRAPPA [34] was used as the software benchmark. It is a standard and widely used program for MP phylogenetic analysis. To achieve its best performance, GRAPPA was run in its *multithreaded mode* on a quad-core 2.40 GHz Intel Xeon E5530 processor with 16 GB of RAM. The run-time measured through GRAPPA served as the basis in our speedup calculations. Specifically, speedups reported are calculated as the ratio of GRAPPA run-time over the total execution time on an N-PE NoC. Note that different multithreaded GRAPPA runs were found to yield different output sequences with the same optimum score. Our NoC simulation also outputs a sequence that matches this optimum score.



(a)



(b)

Figure 3-10: Total execution time in hardware for (a) *SynData_73*, *SynData_50* and *SynData_27* and (b) *SynData_10* and *SynData_04*

3.7.2 Results with Synthetic Data

Five synthetic data sets were generated and used as input. Each input consisted of three genomes with 110 genes each such that m=110. Each data set was generated to have a different common subsequence length and hence different divergence.

Pairwise divergence (δ) is given by subtracting the length of the longest common subsequence from m. We have three values of δ for each input. The standard deviation of the pairwise divergences (σ_δ) was normalized by dividing it by the mean (μ_δ) and used as the divergence metric, Δ ($=\sigma_\delta/\mu_\delta$). This metric serves as a measure of the skew among the three genomes and is made to vary across the entire range of possible values, thereby covering the entire range of the possible input spectrum. Low values of Δ indicate that the genomes are equally far apart irrespective of the actual magnitude of the breakpoint distance. A high value of Δ indicates that two genomes are closer to each other than they are to the third. Five synthetic sets of three genomes each were generated such that the values of Δ in these inputs are 0.731, 0.498, 0.274, 0.103 and 0.039 respectively; these inputs were labeled *SynData_73*, *SynData_50*, *SynData_27*, *SynData_10* and *SynData_04*, respectively. It is also to be noted that the δ values and μ_δ increase as we move from *SynData_73* to *SynData_04*.

3.7.2.1 Timing Performance

Figs. 3.10(a) and 3.10(b) show the total execution times for NoCs with system sizes (N) 4, 16 and 64 for all the synthetic inputs. The total execution time includes the total computation and communication cycles spent in the NoC and the time required to load the data on the NoC using PCIe. It is interesting to note that the absolute run-times are heavily dependent on the input data and the absolute divergences. Since the execution times are a function of the bottleneck number of reductions carried out by the PEs (see 3.6), the execution times for *SynData_10* and *SynData_04* are orders of magnitude higher than

those for the other three inputs. This is because of their larger absolute divergences and hence larger number of reductions performed by each PE. There is not much difference in the run-times on mesh and quad-tree. This is because quad-tree helps reduce only the write latency (as shown in Table 3), which contributes a small fraction to the total execution time in this case.

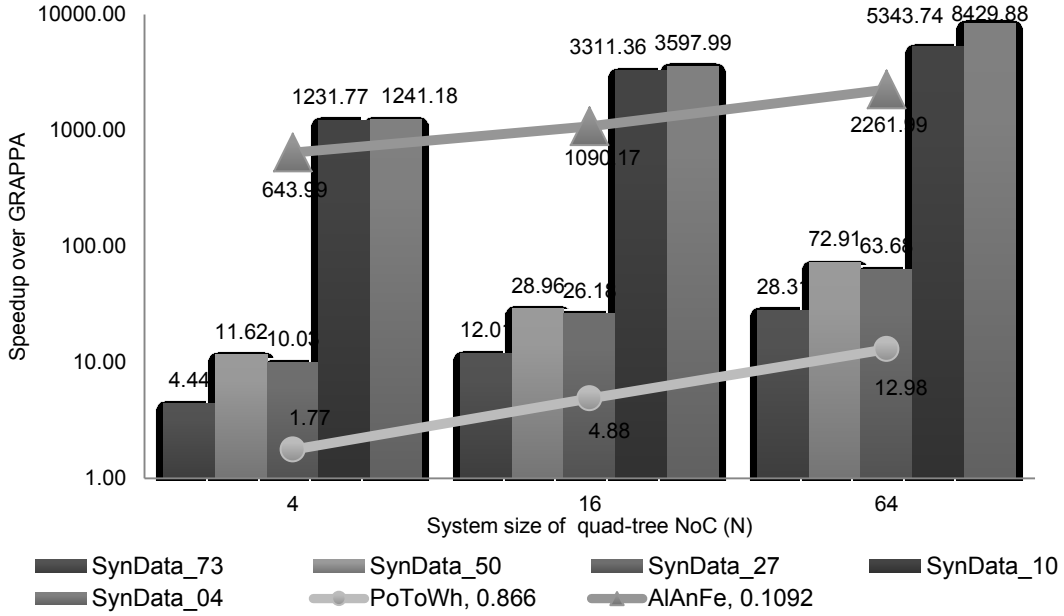


Figure 3-11: Absolute speedup over GRAPPA

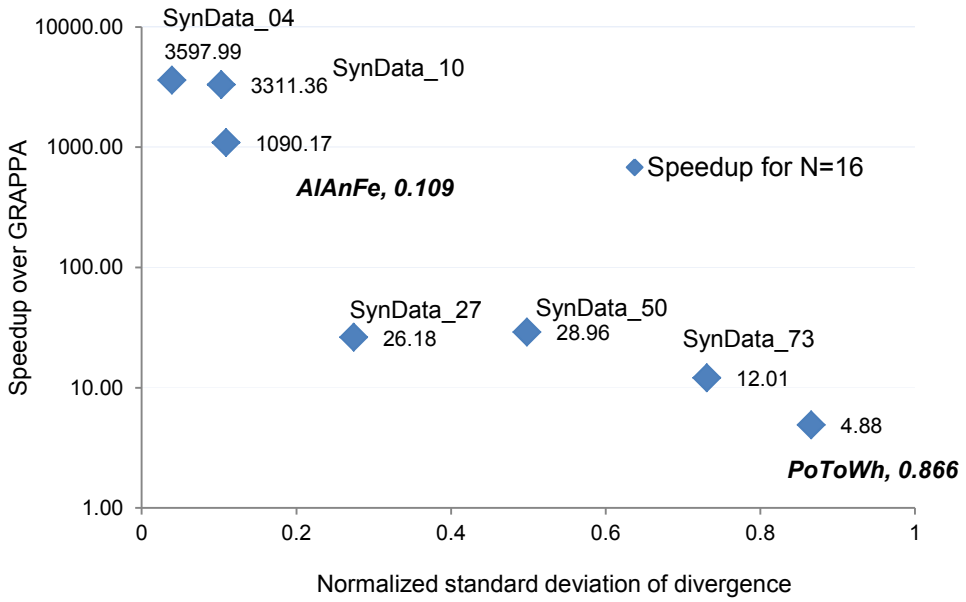


Figure 3-12: Variation of speedup with skew of input data on quad-tree NoC with *N*=16

Fig. 3.11 shows the speedup over GRAPPA using a quad-tree for these inputs. Since speedup is the ratio of GRAPPA's run-time to the execution time on our design, the trends in speedup and execution time are not identical across different inputs. For example, even though execution time increases from *SynData_10* to *SynData_04* for all system sizes, speedup is also observed to increase because GRAPPA's run time increases by a larger factor. Speedup is also dependent on Δ, which indicates that our design is able to accelerate median computation of genomes that are almost equally far apart (e.g., *SynData_04*) significantly more compared to the case where two of the genomes are very close to each other (e.g., *SynData_73*). This observation is more clearly demonstrated in Fig. 3.12, where the speedup on a quad-tree NoC with N=16 is plotted against values of Δ. The best speedups of 1,241 (N=4), 3,598 (N=16) and 8,430 (N=64) are consistently obtained with *SynData_04*. Our results compare favorably with the overall speedup of 417 or the application speedup of 1005 achieved by accelerating GRAPPA in [14].

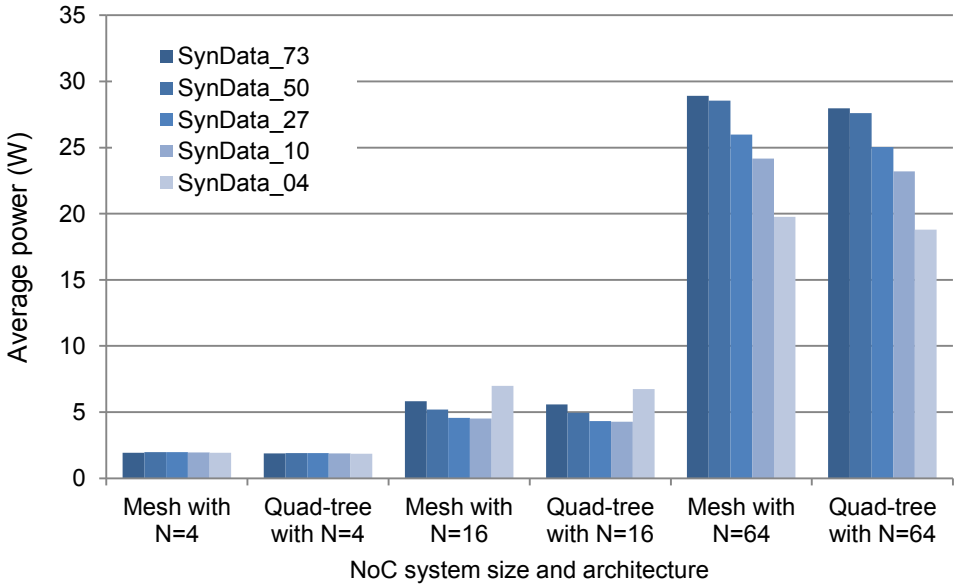
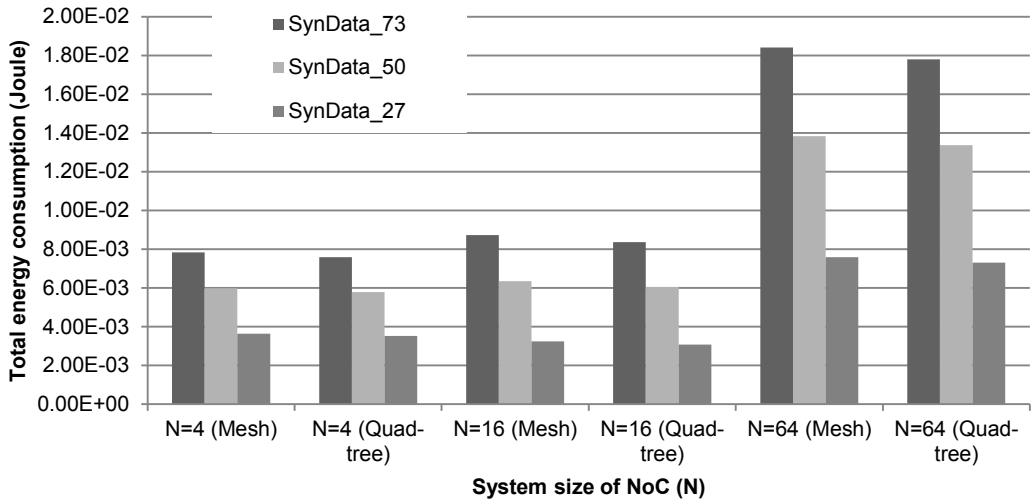


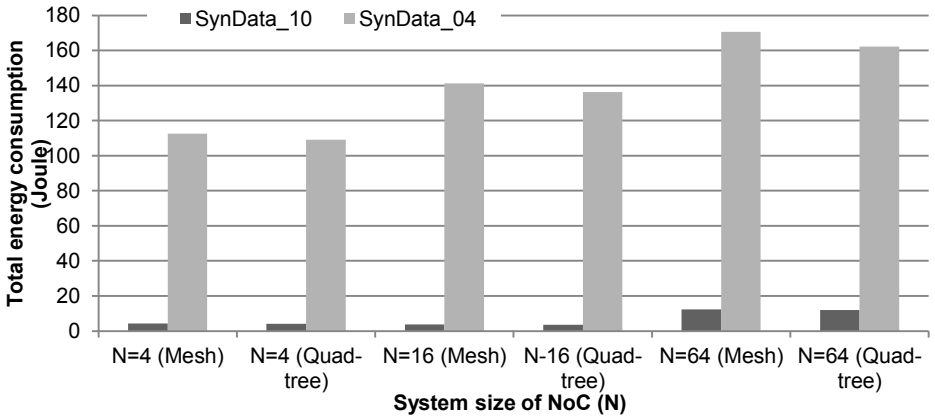
Figure 3-13: Power consumption across various inputs, network architectures and system sizes

Note that the synthetic data encompass almost the full range of possible inputs, with Δ varying from 0.039 to 0.731. Biological inputs can lie on either end of the spectrum or anywhere in between. In particular, as we mention again in 3.7.3, the two real genomic inputs that we use have Δ values of 0.866 and 0.1092. It is also interesting to note that we achieve significantly higher speedups in the cases of genomes displaying greater absolute divergence (*SynData_10* and *SynData_04*). These are also the cases where even highly optimized software implementations such as GRAPPA take very long times to complete. Our design provides better speedup when there is a greater requirement and hence will be of more practical value.

3.7.2.2 Energy Performance



(a)



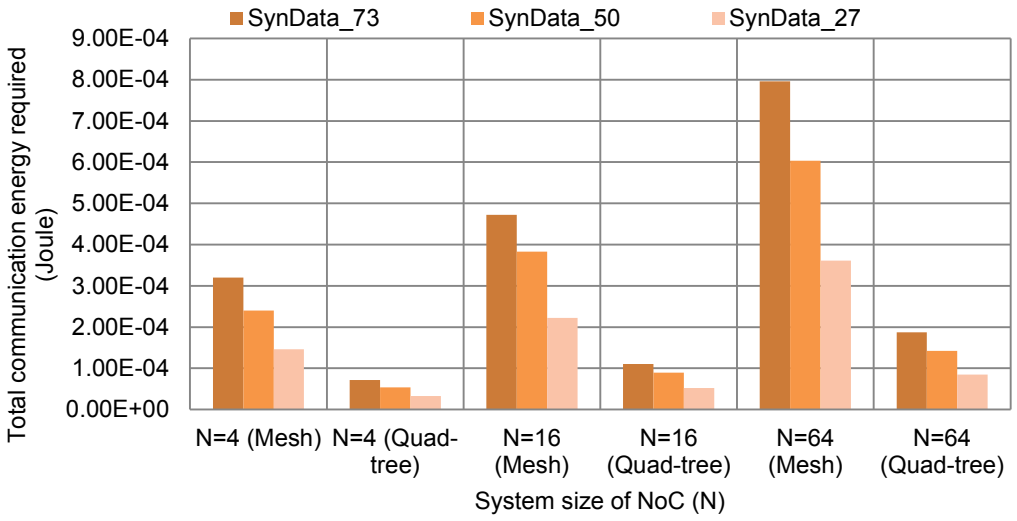
(b)

Figure 3-14: Energy consumption across different synthetic inputs

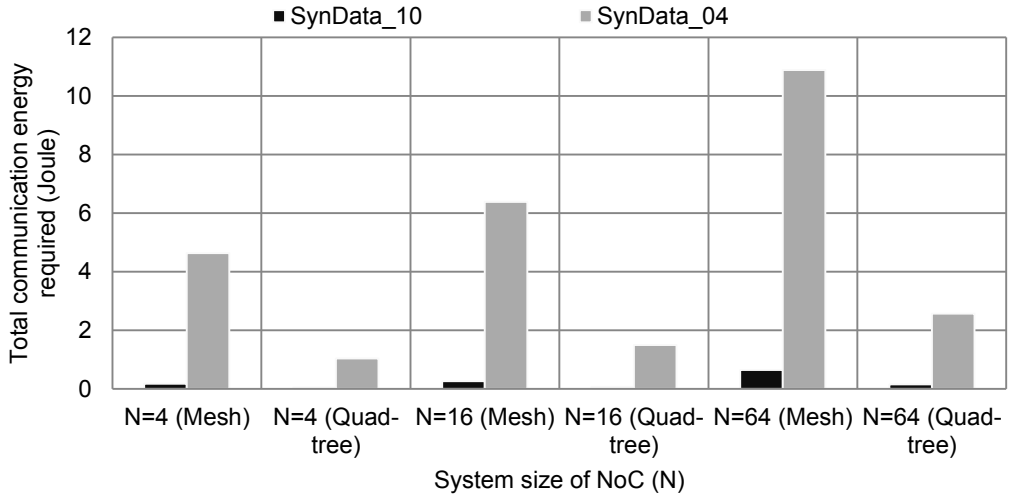
Several measures were used to evaluate the energy performance of the NoC. The average power consumption for mesh and quad-tree NoCs for N=4, 16 and 64 is shown in Fig. 3.13. It will again be noticed that power consumption is a function of the input data, especially for N=64. There is a slight advantage of quad-tree over mesh in terms of power efficiency. For example, a quad-tree NoC consumes up to 5% less power than that based on a mesh NoC. Note that the PEs in both configurations have the same power consumption and the savings

come entirely from the communication architecture. Higher levels of network activity would lead to greater power savings in the quad-tree. However, since the execution time varies widely across inputs, only power consumption provides a partial picture.

A more accurate rubric is the total energy consumption, shown in Figs.



(a)



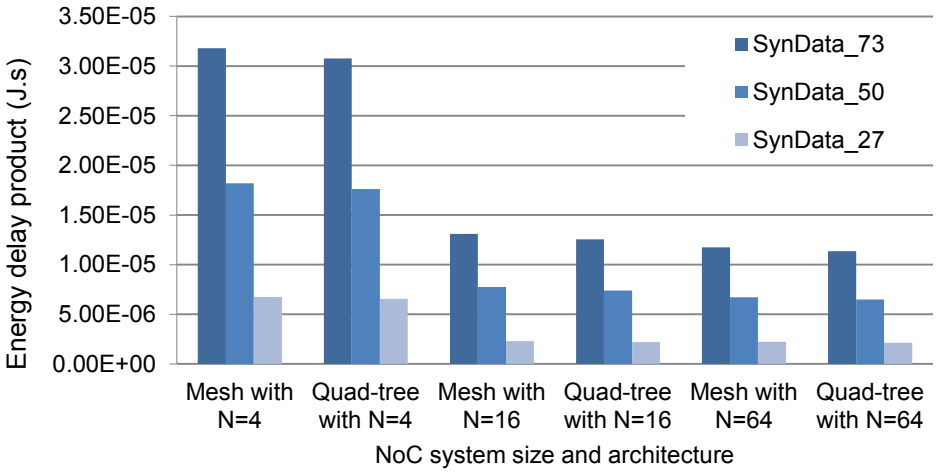
(b)

Figure 3-15: Communication energy expended across different inputs

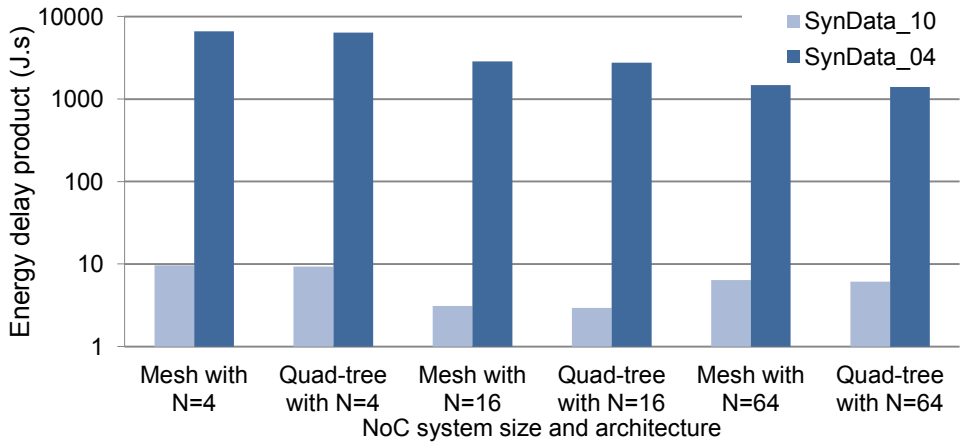
3.14(a) and 3.14(b). Although these figures show the advantage of quad-tree over

mesh in terms of energy performance, comparing only the communication energy consumptions in Figs. 3.15(a) and 3.15(b) further highlights this. Quad-tree consistently outperforms mesh by consuming around 75% less communication energy. Both average power and total energy are input-dependent and generally show a marked increase with increase in system size (N).

The most interesting observation on energy efficiency, however, can be



(a)



(b)

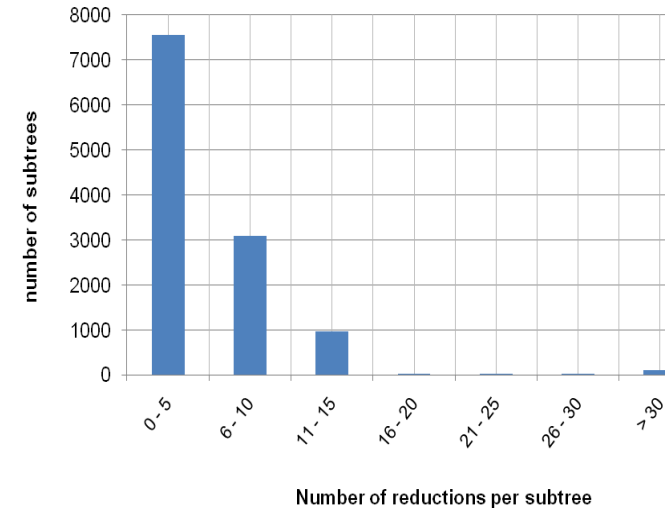
Figure 3-16: Variation of energy-delay product across inputs

seen from Figs. 3.16(a) and 3.16(b) that show the variation of the energy-delay product (EDP) with system size (N) across all inputs. EDP is observed to

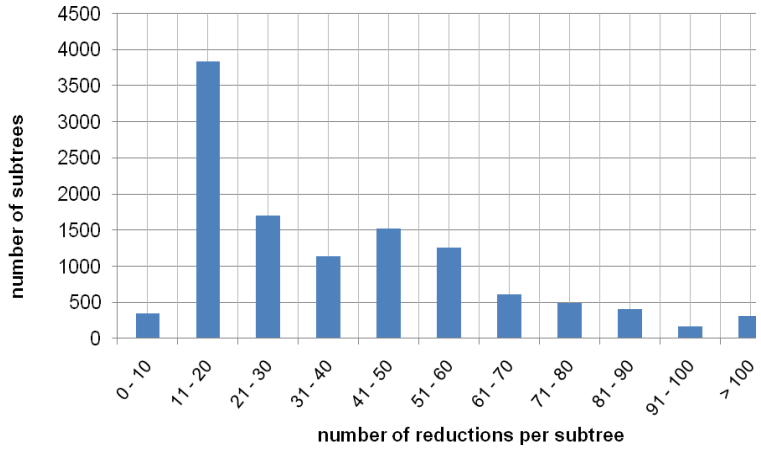
decrease with increasing system size for most inputs. This is because the increase in energy consumption is compensated by the run-time reduction, thereby showing that parallelization is indeed energy-efficient in this case.

3.7.3 Results with Real Genomic Data

Two real genomic inputs were used to evaluate the performance on biological data. Genomic data were downloaded from the National Center for Biotechnology Information's organellar genome repository [89]. One input (*PoToWh*) consisted of the chloroplast genomes of *Solanum tuberosum* (potato, 141 genes), *Solanum lycopersicum* (tomato, 130 genes) and *Triticum aestivum* (bread wheat, 137 genes). The other input (*AlAnFe*) consisted of chloroplast genomes of *Chlamydomonas reinhardtii* (a unicellular green alga, 109 genes), *Brachypodium distachyon* (purple false brome grass, an angiosperm, 133 genes) and *Adiantum capillus-veneris* (black maidenhair fern, 130 genes). These genomes were preprocessed with Mauve [90] in order to determine the common genes. The values of Δ for the inputs are 0.866 for *PoToWh* and 0.1092 for *AlAnFe*. This is indicative of the fact that *PoToWh* represents a skewed data set, with potato and tomato being much closer to one another than they are to wheat. This is expected, as evolutionarily potato and tomato are closely related and belong to the same genus. On the other hand, *AlAnFe* represents a uniformly divergent scenario. The speedups obtained with these inputs for N=4, 16 and 64 are shown in Fig. 3.11. Fig. 3.12 shows the speedup correlation with synthetic data having similar values of Δ.



(a)



(b)

Figure 3-17: Histogram of number of number of reductions per subtree for (a) PoToWh and (b) AlAnFe

As mentioned in 3.7.1, speedup is calculated as multithreaded GRAPPA run-time divided by the total execution time on the NoC. As explained earlier, the total execution time on NoC is proportional to the bottleneck number of reductions. For example with N=16, the bottleneck number of reductions for *PoToWh* is 6,286 and that for *AlAnFe* is 46,958. The total execution times on a quad-tree NoC are 1.14 ms and 8.46 ms respectively. In comparison, the GRAPPA run-times are 5.55 ms and 9.22 s respectively.

Table 4: Reduction statistics for *PoToWh* and *AlAnFe*

	N = 4			N = 64		
	Average reductions per PE	**Standard deviation of reductions per PE**	**Max reductions per PE**	**Average reductions per PE**	**Standard deviation of reductions per PE**	**Max reductions per PE**
PoToWh	15672.75	1847.57	17430	1942.73	194.73	2342
AlAnFe	69222	8558.69	79516	19496.84	1700.02	22614

Next, we turn our attention to the variation of speedup with increasing N. It can be seen from Fig. 3.11 that the speedup on *PoToWh* increases from 1.77 to 12.98 as we increase N from 4 to 64. For *AlAnFe*, the speedup increases from 643.99 to 2,261.99. Table 4 shows the mean, standard deviation and the maximum (bottleneck) number of reductions per PE for *PoToWh* and *AlAnFe*. It is evident that speedup is inversely proportional to the maximum number of reductions per PE. Speedup also varies inversely as the average number of reductions when load is balanced among PEs.

Finally, in order to investigate the reason behind the widely different speedups obtained with *PoToWh* and *AlAnFe*, we plot histograms (Figs. 3.17(a) and 3.17(b)) of the number of reductions per subtree for each of the inputs. The larger skew (Δ) for *PoToWh* is evident from a comparison of the two histograms. Due to the higher skew in *PoToWh*, the best cost is obtained quickly and most subtrees are pruned at the initial stage of the operation, leading to few (< 10) reductions per subtree. The lower skew in *AlAnFe* leads to a more gradual update of the best cost and subtrees are pruned to a lesser degree. Since the

reduction load is shared by several subtrees in the latter case, parallelization provides greater speedup.

3.8 Conclusion

To summarize the work done on Maximum Parsimony or breakpoint phylogeny, we have undertaken the design, implementation and performance evaluation of a NoC-based multi-core architecture for accelerating the breakpoint median problem in phylogeny. Our evaluation encompasses a wide spectrum of inputs, including both synthetic and real genomes. We show that the proposed NoC architecture provides a speedup of up to 8,430 with respect to multithreaded GRAPPA software. We also show how the relationship among the input genomes affects the timing performance of our design and that we are able to provide greater speedup when software methods incur a huge run-time penalty. On the network architecture front, we demonstrate the superiority of a quad-tree over a mesh in terms of energy efficiency for this application class.

We believe that our current implementation provides appreciable performance enhancement over comparable hardware accelerators targeting breakpoint phylogeny, and can serve as a basis for more NoC-based platforms with applications to life sciences. In addition, our design provides a paradigm for accelerating similar vector or matrix-based applications like image processing.

4. NoC-Based Accelerator for Maximum Likelihood

Given models of evolution [6], [7], [8], [9], [10], we can use standard statistical methods to carry out phylogenetic inferences. Their widespread usage is due to the fact that they provide a likelihood score for each reconstructed tree using the PLF [11], [12]. Maximum Likelihood (ML), invented by R. A. Fisher [91], is the most widely-used of such methods. Its application to phylogenetic inference was introduced in [92] for gene-frequency data. ML methods were applied to molecular sequences in [93], [94]. Practical use of ML methods for nucleotide sequences was demonstrated in [11], [12].

The improved quality of result using ML comes associated with a high computational cost as the ML formulation is NP-Hard [13] and suffers from the need to explore a super-exponential (in k, where k is the number of taxa) number of trees. For example, a run using RAxML [70], which is one of the most widely-used programs to compute ML-based phylogeny, on an input comprising of 1,500 genes can take up to 2.25 million CPU hours [69]. As detailed in Chapter 2, prior work done on hardware accelerators for ML have focused on GPU, CBE, FPGA and general-purpose multicores. We propose a NoC-based platform that delivers orders of magnitude speedup over existing methods, and to the best of our knowledge, is the first comprehensive NoC-based solution.

The performance improvements due to the architectural advantages of NoC can be significantly enhanced if 3D integration is adopted as the basic fabrication methodology. The amalgamation of two emerging paradigms – NoCs in a 3D IC environment – allows for the creation of new structures that enable

significant performance enhancements over traditional solutions. The major contributions in this chapter are as follows:

(i) A homogeneous, unified PE design for parallel execution of ML function kernels;

(ii) An efficient, fine-grained implementation of the different floating-point arithmetic operations involved in this application;

(iii) Novel dynamic core-allocation schemes to minimize inter-node communication latency; and

(iv) Exploration and evaluation of the merits of different 2D and 3D NoC architectures.

We demonstrate the capability of our NoC-based platforms to achieve function-level speedups of 390x to 847x, aggregate speedups of accelerated kernels in excess of 6500x, and end-to-end run-time reductions of over 5x with respect to state-of-the-art multithreaded software.

4.1 Theoretical Background

Likelihood is of central importance in statistics. From Bayes' Theorem, given a hypothesis H and observation D, we have the following:

$$P(H|D) = \frac{P(H \cap D)}{P(D)} = \frac{P(D|H) * P(H)}{P(D)} \qquad (3)$$

Here, P(H | D) is the a posteriori probability of H, P(H) is the a priori probability of H and P(D | H) is the likelihood of H. Given two hypotheses H_1 and H_2 and n independent observations D_1, D_2, …, D_n constituting D, we can express the odds ratio in favor of H_1 over H_2 as follows.

$$\frac{P(H_1|D)}{P(H_2|D)} = \left(\prod_{i=1}^{n} \frac{P(D_i|H_1)}{P(D_i|H_2)}\right) * \frac{P(H_1)}{P(H_2)} \qquad (4)$$

From (4), we can see that the likelihood ratio dominates the right hand term for a large number of observations (data). Bayesian statisticians try to come up with valid a priori probabilities and use Bayes' Theorem to infer valid a posteriori probabilities for their hypotheses. Non-Bayesians prefer the hypothesis that maximizes the likelihood P(D | H). For a large amount of data, this also turns out to be the hypothesis with the largest a posteriori probability P(H | D) and hence the best estimate.

We now touch upon some basic features of the method of using likelihood for phylogenetic tree computation. Details are provided in Chapter 16 of [12]. Initially, we have a set of aligned DNA sequences with m sites (or columns). Several of these sites are identical in their nucleotide composition. A group of adjacent sites that are equivalent is replaced by a single site with a "weight" indicating the multiplicity of that site. Let the number of sites after this compression be m'. A given phylogeny (or phylogenetic tree) consists of branch lengths and a model of evolution that allows us to compute the probabilities of state changes along this tree, in particular, the probability $P_{ij}(t)$ of state i transitioning to state j at the end of a branch of length t. The following assumptions greatly simplify the process of computing likelihoods.

(i) Evolution in different sites (on the given tree) is independent.

(ii) Evolution in different lineages is independent.

Assumption (i) renders likelihood computations simple by focusing on one site. Likelihood values for the entire sequence can be found by multiplying the

likelihood values for each site. Complex models incorporate rate variation across sites using a hidden Markov model (HMM). One of the most common models [95] uses autocorrelated gamma distribution approximated by having discrete categories. Assumption (ii) allows us to write the conditional probability at each level only with respect to its immediate predecessor and the intervening branch length. These assumptions enable us to calculate the likelihood of a tree using a bottom-up approach, starting from the observable data to the "root" of the tree. It is further shown in [12] that the phylogenetic tree is unrooted and the placement of the root is important only when we assume molecular clocks.

4.2 Existing Software Suites for ML Phylogeny

PAUP* Version 4.0 [96] is an improvement on previous versions of PAUP: Phylogenetic Analysis Using Parsimony and is the most widely-used software package for the inference of evolutionary trees. It is a general-purpose package that combines parsimony, distance matrix, invariants and maximum likelihood methods, and many indices and statistical tests. PHYLIP [97] (Phylogeny Inference Package) is one the oldest distributed packages that includes ML as one of the methods and can operate on data types including molecular sequences, gene frequencies, restriction sites and fragments, distance matrices and discrete characters. PHYML [98] is a software that implements a fast and accurate heuristic for estimating ML phylogenies from DNA and protein sequences. The tool provides the user with a number of options, such as nonparametric bootstrap and estimation of various evolutionary parameters, in order to perform comprehensive phylogenetic analyses on large datasets in run-times comparable

to parsimony programs. Other programs like fastDNAML [99] render faster solutions for larger trees and number of species.

RAxML [70] provides a very fast reconstruction of phylogenies using ML. RAxML 7.0 offers several ways to exploit parallelism, in addition to its sequential version. It is a highly optimized program that handles DNA and amino acid alignments under various models of substitution and several distinct methods of rate heterogeneity. In addition, it has a novel rapid bootstrapping algorithm built into it, which when combined with rapid ML search allows users to conduct a full ML analysis in a single program run. RAxML is able to handle extremely large data sets as shown in [100]. We chose RAxML as the ideal candidate for which to explore hardware acceleration possibilities because it is the most optimized and parallelized software currently available. As mentioned in Chapter 2, several of the existing papers on hardware acceleration for ML phylogeny target RAxML.

4.3 Design of Computation Core

The motivation for using a NoC to address the ML application stems from the fact that there are different levels of parallelism in the application that can be exploited to accelerate the computation. Fine-grained parallelism can be exploited within a processing element (PE) to render a fast hardware implementation for each phylogenetic function kernel. While the same can also be alternatively implemented on a large FPGA board that supports several computation cores (e.g., similar to [72]), a NoC based multi-core system can handle coarse-grained parallelism more efficiently. The latter requirement

becomes particularly important in the context of ML programs because they typically involve a large number of function invocations; and at any point of execution there could be variable number of instances running for each function. Under the NoC framework, these requirements can be effectively addressed by

(a) Designing a homogeneous system, where different PEs are able to seamlessly support different functions executing at different times, and

(b) Interconnecting them using a suitable network that allows concurrent execution of arbitrary combinations of function instances and provides the backbone for efficient data exchange between the individual PEs.

In other words, we can build a heterogeneous application map on a homogeneous-core NoC. Furthermore, such a homogeneous NoC-based system can be allowed to scale up to provide the computation bandwidth necessary for solving larger problems.

The computation of ML phylogenetic kernels requires the use of elementary functions, specifically logarithms and antilogarithms, in addition to basic arithmetic functions. Fast calculation of logarithms in hardware has been a well-researched topic. Kwon et al [80] describe a fast implementation of exponentiation in hardware targeting graphics applications. A 32-bit binary-to-binary linear approximation-based logarithm converter is described in [81]. Optimality of Chebyshev polynomials for table-based approximations of elementary functions is described in [82]. A unified computation architecture for calculating elementary functions, including logarithm, exponential and multiply-and-add, is presented in [83]. They use a fixed-point hybrid number system

(FXP-HNS) to integrate all operations in a power and area-efficient manner with a low percentage of error. They achieve a throughput of 1 data output per 4.3 ns cycle for elementary functions. Another technique for designing piecewise polynomial interpolators for implementing elementary functions in hardware is described in [84]. They designed linear, quadratic and cubic interpolators with progressively increasing accuracy for both high speed and low power. Our design builds upon the method employed in [83].

4.3.1 PE Design

The PE aims to capture the crux of the computation involved in phylogenetic kernels. We address this by combining fast and efficient computation strategies in hardware with extensive fine-grained parallelism. The

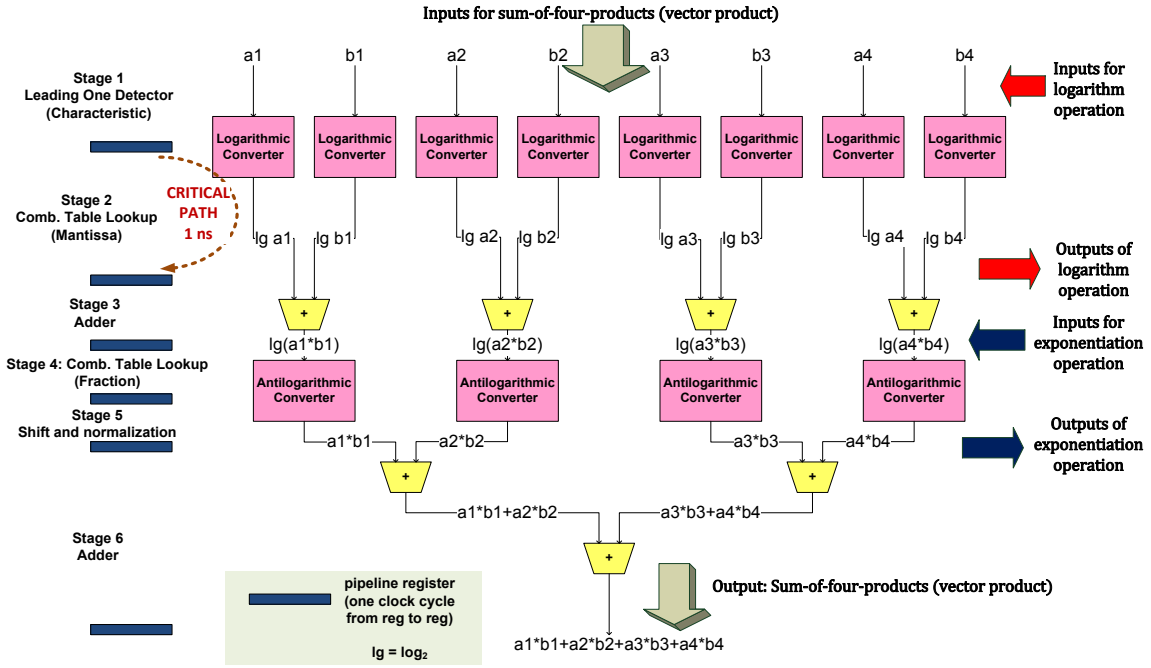


Figure 4-1: Architecture of computational core for sum-of-products, logarithm and antilogarithm

core architecture uses FXP-HNS arithmetic [83] and has six pipeline stages as shown in Fig. 4.1.

It can compute a sum of four products during one traversal across the stages, in addition to regular logarithm and exponential. Logarithm and exponential are directly computed using linear table-based approximations as in [83]. These functions are entirely implemented using logic gates, without using a ROM. For sum of products, an indirect approach is used. Logarithms of four pairs of numbers are taken, each log value in a pair is added to the other, four antilogarithms are taken and the four results are added together. In other words, multiplication is done by addition in the log-domain and addition is done in the linear domain. Figure 3.18 shows a schematic diagram of the computation architecture. Note that if we are interested in only computing the logarithm, the adder stage is not required and the core provides the result of the stage 2 as output depending on the instruction being executed. For computing exponential, the input goes directly to stage 4 with minor modifications in the number representation format and the output is available at the end of stage 5. The three representative functions of the RAxML suite that are used, namely *coreGTRCAT*, *newviewGTRCAT* and *newviewGTRGAMMA* [101] are instruction-coded to be run on the computation core. The core is instantiated within a wrapper that provides instruction decoding, data fetching and data write-back functions. The design has been implemented with Verilog HDL and synthesized with a clock frequency of 1 GHz using 65 nm standard cell libraries from CMP [88]. The critical path delay is 1 ns as shown in Fig. 4.1.

4.3.1.1 Memory Subsystem

The computation core has the requisite memory to store the input vectors and the computation results for each step of the function computation, all in FXP-HNS format. The per-PE memory requirement is 0.5 MB. This is implemented in the form of register banks. As mentioned earlier, there are no ROM-based lookup tables for computing logarithm and antilogarithm.

4.3.2 Automating Column Compression in Hardware

As mentioned earlier and also in [102], the cost of the likelihood function and the branch length optimization function, which accounts for the greatest portion of the execution time can be reduced by (a) reducing the search space using some additional heuristics, and (b) reducing the number of sites taken into

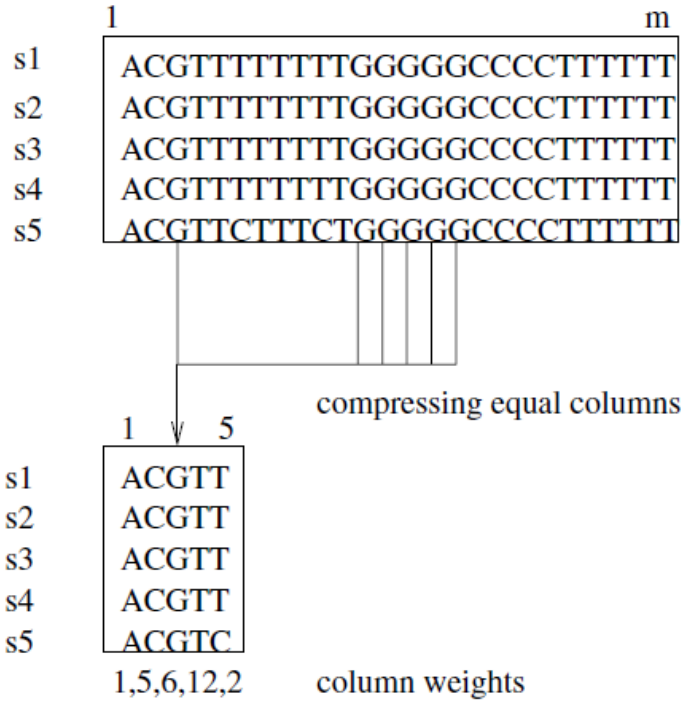


Figure 4-2: Global compression of equal columns for five input sequences. Note that 26 columns are compressed to 5 with appropriate weights assigned to each.

account during computation, thereby reducing the number of computations at each inner node during the evaluation of a tree. The algorithm used in [102] for determining column equalities and compressing equivalent columns is mentioned here for the sake of convenience.

4.3.2.1 Algorithm

Two columns in an alignment are equal and belong to the same *column class*, if the base is the same on a sequence-by-sequence basis. A column is *homogeneous* if the same base exists across all sequences and *heterogeneous* otherwise.

Let s_1, s_2, ..., s_n be the set of aligned input sequences as depicted in the upper matrix of Fig. 3.19. Let m be the number of sequence positions of the alignment. Two columns, i and j, of the input data set are said to be equal if $s_{ki} = s_{kj}$ for all $k = 1 \ldots n$. It is now possible to calculate the number of equivalent columns in a column class and compress the columns in the input data accordingly. The compression is carried out by replacing all columns belonging to one column class by one representative column and assigning a weight to that column denoting the original number of columns in that column class. The number of columns after column compression is denoted by m'. Referring to Fig. 3.19, for example, $m = 26$ and $m' = 5$. Since phylogenetic tree reconstruction is preceded by a high quality multiple sequence alignment, a large number of column equalities are expected at the global level. This leads to a great deal of compression so that m' is usually much less than m.

Another level of compressing the input data for phylogenetic tree reconstruction lies in identification of homogeneous or heterogeneous columns. In case of homogeneous columns, the tree for that site need consider only one base irrespective of the number of sequences. This aids further compression of the input space. Referring to Fig. 3.19 again, we see that 4 out 5 columns in the column-compressed input are homogeneous.

4.3.2.2 Design

From the algorithm described in 3.2.3.2.1, it is evident that that time complexity of the design is *O(m)* and the space complexity is *O(mn)*. One column

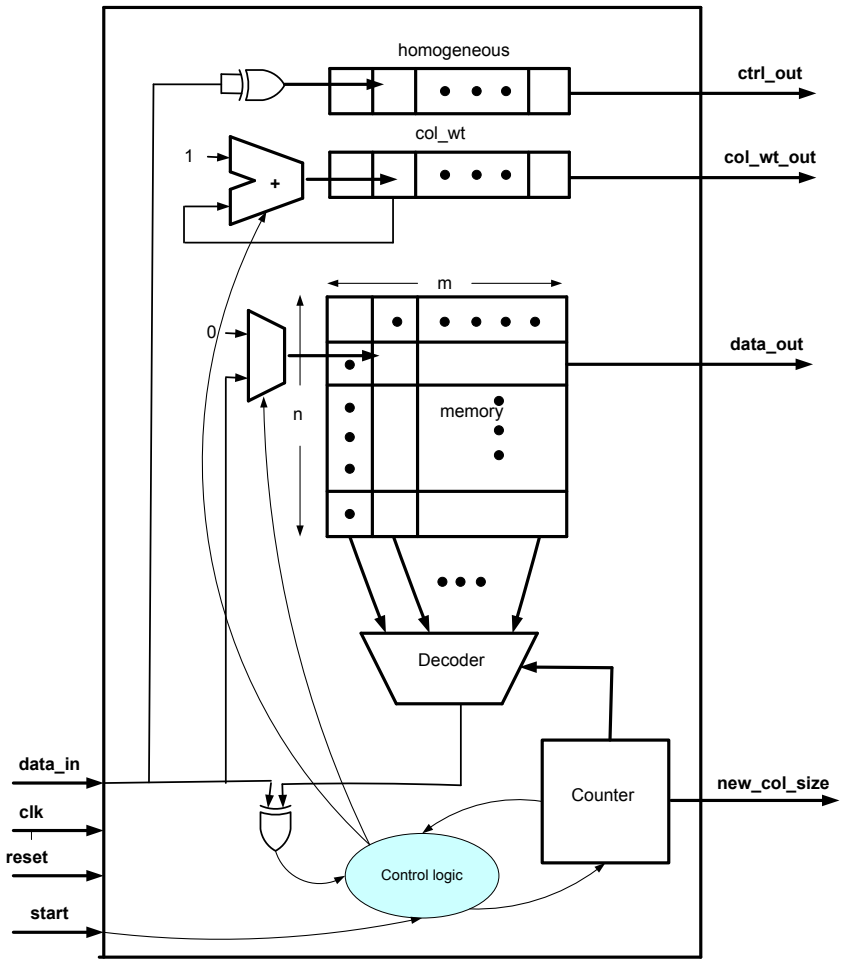


Figure 4-3: Schematic diagram of Column Compressor

from each sequence is taken as input in every cycle and is stored in an internal memory after column processing. An external control signal determines whether the processor is in input or output mode. The number of cycles spent in input mode is *m* and the number of cycles spent in output mode is *m'*. Since processing is concurrent with input, no extra cycles are needed.

The schematic diagram of the design is shown in Fig. 3.20. The signal *start* switches between input and output modes. The internal *memory* stores the sequences after column equalities have been determined. The column weights are stored in *col_wt*. The *homogeneous* registers store a 1 if the column is homogeneous and 0 otherwise. The maximum value of the counter is used to determine *m'*, which goes to *new_col_size* as output. Outputs are activated once *start* goes low.

4.4 NoC Node

The core with a wrapper is designated as a *processing element* (*PE*). Four

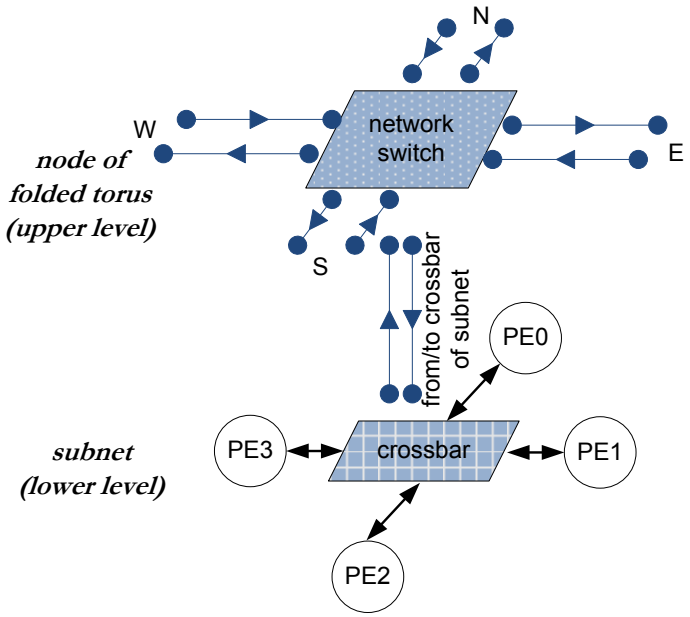


Figure 4-4: Network switch of NoC and cross-connected subnet under one node

such PEs are integrated to form one subgroup. The choice of this subgroup size comes from the fact that the three functions require different numbers of sum-of-four-product computations (8, 12, 24) for which the greatest common divisor is four. The PEs are labeled *PE0*, *PE1*, *PE2* and *PE3*. A crossbar switch, shown in Fig. 4.4, connects the four PEs and coordinates communication among them. The crossbar switch has to deal with three kinds of traffic, which consists of intermediate function results. The first and simplest kind involves sending data received from *PEx* back to *PEx*. The second kind of communication involves sending data from one particular PE to all the other three PEs within the subgroup. The third kind of communication involves sending/receiving data to/from the external network through a network switch. This subgroup of four PEs along with the crossbar switch forms a *subnet* under one *NoC node*. The number of nodes in the system is denoted by *N*.

4.5 Network Architecture

The choice of the network is determined by the traffic patterns [103] generated by the application. In our case, a single RAxML run typically generates millions of invocations of a few functions at different time-points, and each of these functions can benefit from fine-grain parallelism by an assignment to multiple PEs. This leads to a *high volume of arbitrary point-to-point communication*. In addition, we observed dynamically changing traffic patterns and a clear absence of steady-state localized traffic or clustering, all of which indicate the desirability of a distributed interconnection topology. A statistical analysis of the traffic patterns under the assumption of an underlying folded torus network reveals this fact. The mean and normalized standard deviation of

the number of flits per cycle contained in the buffers of each router in an 8×8 folded torus for a typical application scenario is shown in Fig. 4.5. The clear lack of clustering can be observed from the absence of prominent peaks in the mean traffic plot in Fig. 4.5(a). The dynamically varying nature of the traffic can be gleaned from Fig. 4.5(b) that shows substantial standard deviation of traffic (typically above 50% of the mean) across simulation cycles. Hence, topologies like

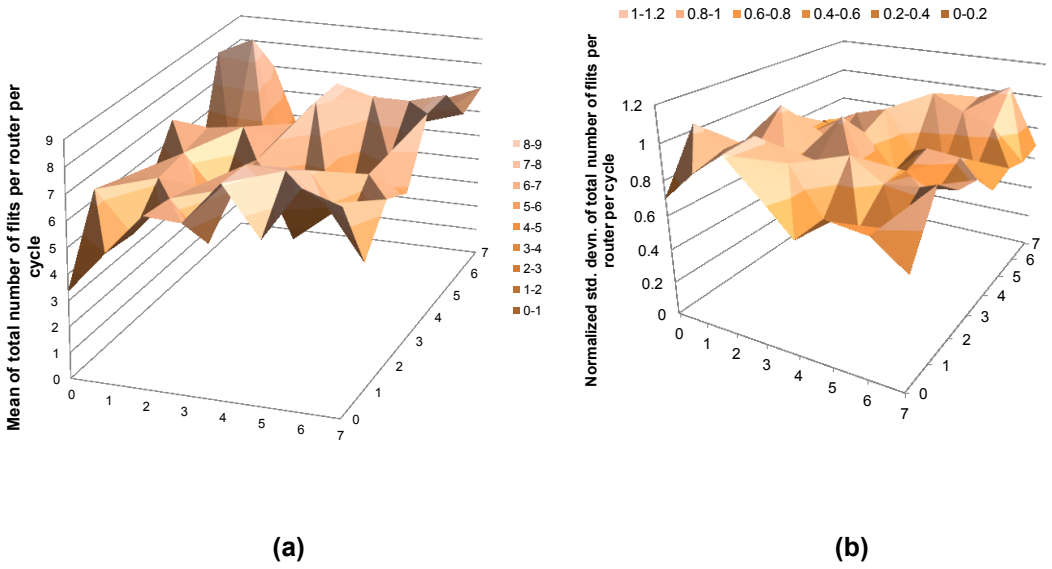


Figure 4-5.Mean (a) and normalized standard deviation (b) of flits per cycle in routers in a folded torus network

star or quad-tree that cater to regular or localized traffic patterns would not benefit this application scenario.

From the VLSI implementation perspective, a mesh is a scalable network architecture whose regularity provides for easier timing closure and reduces dependence on interconnect scalability [27]. A folded torus further reduces the *point-to-point separation* (Manhattan distance) between nodes by cutting down the diameter of the network by half without compromising on the regularity or

scalability of the entire network. Hence, we decided to explore folded torus in our 2D NoC design.

Three-dimensional ICs that contain multiple layers offer advantages like reduced length of interconnect, higher package density, lower power consumption and higher noise immunity [74]. 3D ICs can be used to improve performance by forming a processor-memory stack, as shown in [75]. This enables use of very wide buses and stacks to drastically reduce memory access time. 2D mesh structures are compared with their 3D counterparts in [76] by analyzing zero-load latency and power consumption of each network. A more detailed evaluation that takes into account various real-world traffic patterns and carries out cycle-accurate simulations is presented in [77]. 3D NoCs have been proposed for improving the performance of application-specific architectures in [78]. 3D design-space exploration for cache memories has been considered in [79].

3D NoCs provide enhanced performance due to the additional degree of freedom in the vertical dimension, thereby enabling better integration and reduced inter-node hop-count for larger system sizes [77]. We explored the design of two different 3D NoC architectures: 3D folded torus and 3D stacked torus; and used a system size N of 64 (=4×4×4) in our application study. A 3D folded torus NoC has a folded torus along each dimension (x, y and z). There are one-hop vertical links (in the z dimension) between adjacent layers. On the other hand, a stacked torus [79] is a hybrid between a 2D folded torus, which is a packet-switched network, and a bus, which takes advantage of the short inter-layer

distances. It integrates multiple layers of folded tori by connecting them with buses spanning the entire vertical height of the chip. Hence any inter-layer communication (for the same <x,y> coordinates) is one-hop.

Since each subnet associated with a node has 4 PEs, our system has 64 PEs for N=16 and 256 PEs for N=64. A network switch handles traffic emanating from or destined to each network node. We use the switches described in [104] and [77] for our design. Each switch in the 2D architecture has four bidirectional ports to neighboring switches and one bidirectional port to the crossbar switch of the subnet (Fig. 4.4). In the 3D folded torus, the switch has two additional ports (total 7 ports) to the layers above and below. Alternatively, in the 3D stacked torus, the switch has just one additional port (total 6 ports) to the vertical bus connecting all the layers.

We adopted wormhole routing-based data exchange among the NoC nodes. The primary data contained in the messages exchanged among nodes are intermediate function results, which are 64-bit numbers using FXP-HNS format [83]. Given this small message size, we split each message into 3 flits (header, body and tail), each of width 64 bits. Since deeper buffers may slow down clock frequency and do not appreciably improve performance for short messages [105], we use buffer depth of 2 flits. We adopt the routing and arbitration mechanism from [104] and [77].

4.6 Function-Level Parallelization

The three target phylogenetic function kernels from RAxML are *newviewGTRCAT* (*f2*), *coreGTRCAT* (*f3*) and *newviewGTRGAMMA* (*f6*). We parallelize each function by breaking down larger computation arrays into smaller units as follows: Taking *newviewGTRCAT* (*f2*) as an example, we can see from Fig. 4.6 that computation of the *x3* array in each iteration requires computation of eight sums-of-four-products, using arrays *left*, *x1*, *x2* and the eigenvalue vector *EV[15:0]*. Since each PE can compute one sum-of-four-products, eight PEs (or equivalently, two NoC nodes) are required. We refer to this function as *f2*, indicating that its computation requires two NoC nodes. Similarly, each iteration within the function *coreGTRCAT*

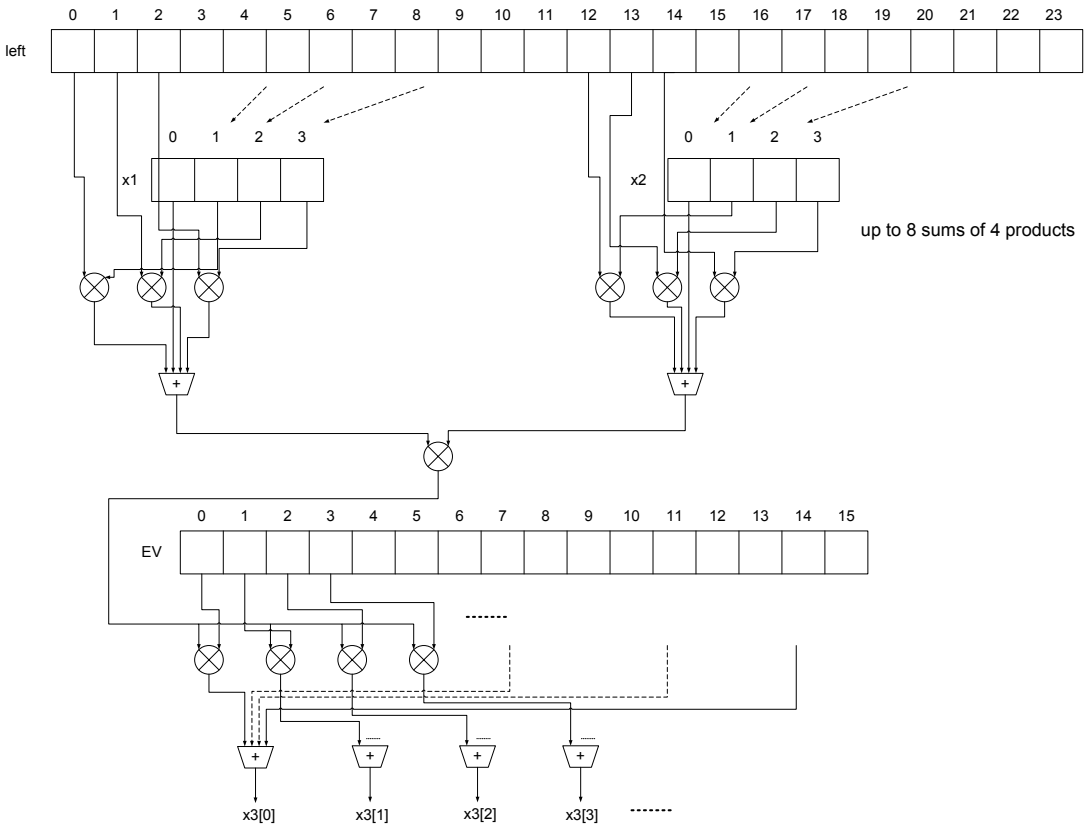


Figure 4-6: Part of the computation tree of *newviewGTRCAT*

(*newviewGTRGAMMA*) involves computation of up to twelve (twenty-four) sums-of-four-products, thereby requiring three (six) NoC nodes. Hence, we refer to it as

f3 (*f6*). Other operations, such as carrying out the exponentiation operation involved in computing the *left* vector in *newviewGTRCAT*, are also computed in the PEs within each node. Also, in *coreGTRCAT* for example, sum-of-products, exponentiation and cumulative addition are involved. All these operations are executed in parallel over multiple PEs. Intermediate results are redistributed among PEs within the same node using the intra-node crossbar switch and among other nodes using the network switch/router.

4.7 Dynamic Node Allocation

A node is *busy* when the PEs within its subnet are collectively executing a function; otherwise it is *available*. Nodes continually keep sending their busy/available status to a centralized controller (*MasterController*) that dynamically allocates a subset of nodes from the set of available NoC nodes to a function. If the number of available nodes at any point of time is less than the number of nodes requested by that function (2, 3 or 6), the function waits till the requisite number of nodes is available. The nodes allocated for executing one function instance are said to belong to one *partition*. Nodes can be reused after execution of the function has completed in the partition.

Since the nodes belonging to a given dynamically-allocated partition need to communicate with one another, it is desired that they be co-located on the network in order to reduce the number of hops required for data exchange and thereby reduce the communication latency associated with the function. A good allocation strategy needs to ensure this co-locality, as well as execute fast enough to not introduce any significant allocation overhead.

One approach for allocating a partition is to use breadth-first search (BFS) on the network. Although this appears to be a reasonable strategy, there are certain drawbacks: First, BFS does not guarantee the co-locality of non-root nodes. The dispersion could become greater if the root node for BFS lies in a neighborhood containing a majority of busy nodes. Higher dispersion among allocated nodes in a partition results in a higher average message hop-count, resulting in higher communication latency. Secondly, the allocation overhead becomes dependent on the choice of the BFS root node, and growing a partition

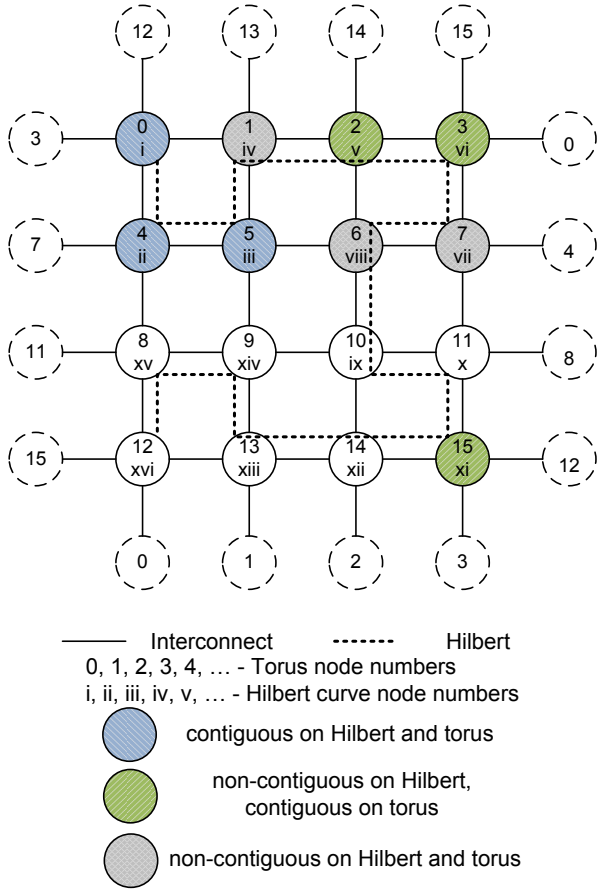


Figure 4-7: Hilbert curve embedded in the folded torus network architecture and different kinds of contiguous and non-contiguous partitions.

around a root node surrounded by a majority of busy nodes has the risk of increasing the allocation overhead. This is because the *MasterController* handling the node allocation has to traverse each node in the neighborhood (in

the adjacency list of the parent node). Scanning the adjacency list of each node takes one clock cycle, and therefore, growing the full partition requires a number of cycles equal to the number of nodes requested by the function in the best case, and N clock cycles in the worst case, where N is the system size.

In this paper, we have developed a novel approach that uses the Hilbert curve [106] for the dynamic allocation problem. A Hilbert curve is a locality-preserving space-filling curve widely used in scientific computing [107]. In addition, this approach results in consistently lower allocation times, as described below. In the following sub-sections, we describe different approaches for dynamic node allocation that make use of the Hilbert curve superposed on different 2D and 3D NoC architectures.

4.7.1 2D Hilbert Curve with Serial Scan and First Fit (*2D_serial*)

In our first approach, we use the Hilbert curve on a 2D folded torus as follows: The *MasterController* serially scans the nodes along the Hilbert curve and chooses the required number of available nodes and allocates them as a partition to the requesting function.

Using the Hilbert curve offers a couple of key advantages. A Hilbert curve has the property that when mapped onto a regular mesh or a folded torus, nodes adjacent along the Hilbert curve traversal are also adjacent on the network. Furthermore, there could be nodes which are not adjacent along a Hilbert curve but are adjacent on the folded torus. Also, a Hilbert curve is essentially converting a two-dimensional allocation problem into a one-dimensional problem. Taking advantage of this property, we use a fixed Hilbert curve

embedded on a folded torus as shown in Fig. 4.7 (for N=16), where there is a one-to-one correspondence between the node ids on the torus and those on the Hilbert curve. This allows us to effectively predetermine the set of possible nodes for allocation. Since this information can be hard-wired in the design, the allocation of an entire partition can be achieved in one clock cycle (for N=16) or four clock cycles (for N=64) in our design.

Note that our Hilbert curve-based approach may lead to three scenarios, as shown in Fig. 4.7. First, allocated nodes are all adjacent to each other or *contiguous* on Hilbert and hence the *partition* is *contiguous* on the torus. Second, the nodes are *non-contiguous* on Hilbert but form a *contiguous partition* on the torus. Third, the nodes are *non-contiguous* both on Hilbert and on the torus.

4.7.2 Multiple 2D Hilbert Curves with Parallel Scan and Best Fit (*2D_parallel*)

Despite the ease of implementing the *2D_serial* approach, there are two main drawbacks. First, there is a constant allocation penalty of 4 cycles per partition for a system size of 64. In addition, the allocation policy in *2D_serial* is first-fit. Hence, it does not guarantee allocation of a contiguous partition even if one is available. Therefore, we developed an alternative approach, *2D_parallel*, where we make the following changes to the allocation policy. This policy is particularly suited for larger system sizes; so we will use N=64 in the following description of the underlying algorithm:

(a) First, we use four Hilbert curves on a square folded torus in *2D_parallel* (instead of one as in *2D_serial*). These four curves are obtained by using right-angle rotation operations of a single Hilbert curve.

(b) We further divide each of the four Hilbert curves into four *segments*, one from each quadrant – thereby resulting in a total of 16 segments. The *MasterController* module now has 16 heads, each of which is responsible for scanning a segment. All 16 heads act in parallel.

(c) Each head now preferentially looks for a contiguous partition starting from any of the nodes in its segment. The first head to find a contiguous partition returns it to the requesting function and interrupts all the other scanning heads.

(d) In case, no contiguous partition is found after each head has finished scanning its segment, we fall back to execute *2D_serial*.

We have experimentally verified that case (d) has a low probability (< 0.2) of occurring and a contiguous partition can be found in most cases. Although there is an additional allocation penalty due to the best-fit strategy we use (step (c)), it provides a higher percentage of contiguous partitions than is obtained using *2D_serial*. The average number of cycles spent per allocation comes down from 4 to 3.22. More importantly, greater contiguity of allocated partitions reduces inter-node communication latency and provides better speedup, as will be shown in Section 4.9.

4.7.3 3D Folded Torus NoC (*3D_torus*)

To further improve contiguity of the allocated partitions while spending fewer allocation cycles, we map our application to a 3D folded torus architecture. The NoC is a 4×4×4 folded torus as shown in Fig. 4.8 (a). A 2D 16-point Hilbert curve is embedded on the top layer (layer 0) and is used to allocate partitions. For each allocation request, *MasterController* allocates all *available* nodes in the column (consisting of 4 layers) corresponding to the current head position. The next head position follows from the 16-point Hilbert curve. This is done till all requested nodes are allocated. In addition, we ensure vertical contiguity by flipping the vertical direction of allocation. For instance, if the most recent node allocated in the current column is from layer 3, the next node to be allocated

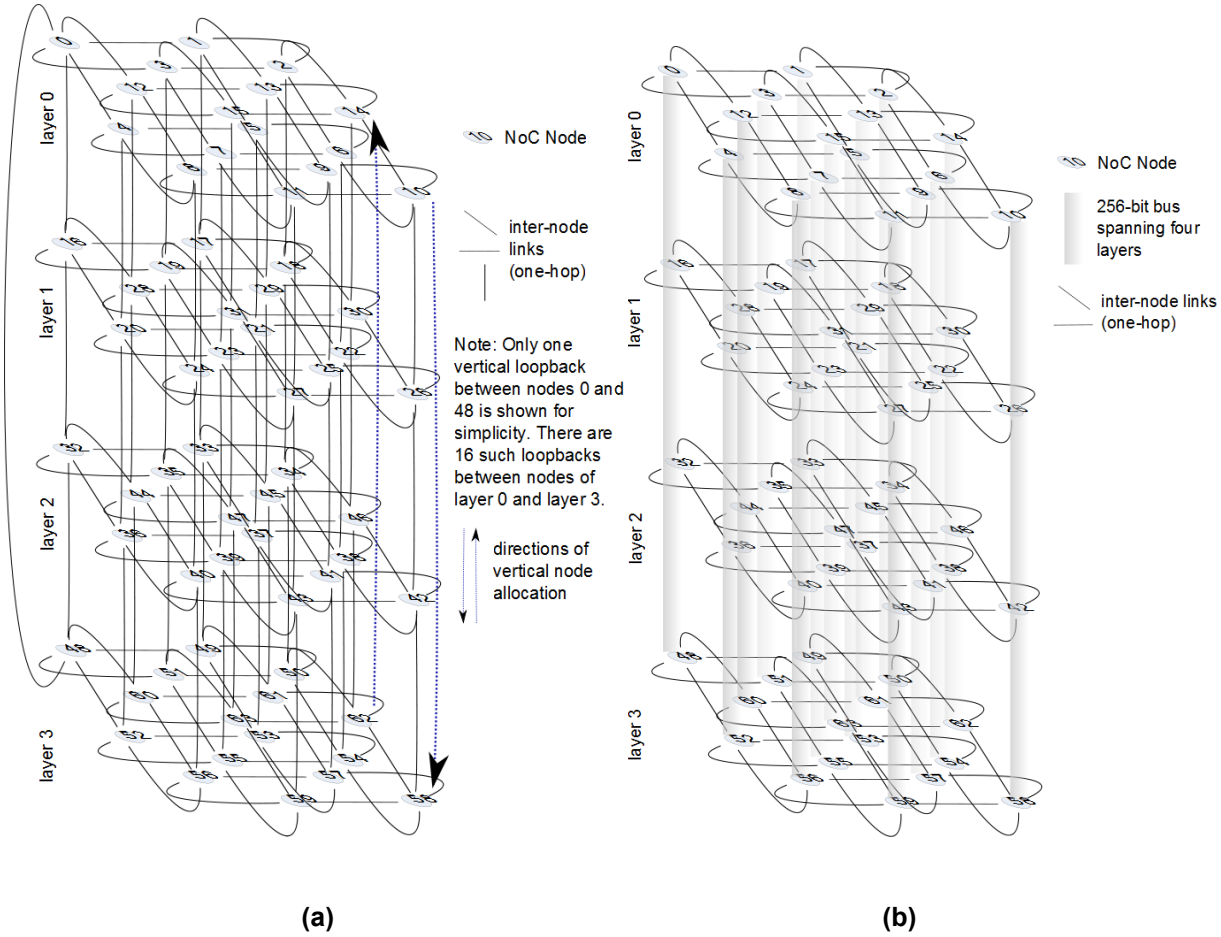


Figure 4-8. (a) 3D folded torus NoC architecture for *N*=64; also shown are the alternating vertical node allocation directions. (b) Stacked torus NoC architecture for *N*=64.

comes from layer 3 in the column corresponding to the next head position. In other words, we alternately move up and down the columns during node allocation. We are able to handle allocation of nodes in one vertical column in one cycle; hence the average allocation time for *3D_torus* goes down to 1.56 cycles. As shown in Section 4.9, *3D_torus* provides the highest speedup and greatest energy efficiency.

4.7.4 3D Stacked Torus (*3D_sttorus*)

Another popular 3D NoC architecture is the stacked torus. For our application, we have four 4×4 folded tori vertically stacked using 16 buses as shown in Fig. 4.8 (b). Bus width is a determinant of the performance of a stacked torus. As shown in [77], a stacked torus with a bus width of 4 flits achieves the same throughput performance as of a 3D torus. Hence, we use a bus width of 4 flits, i.e. 256 bits in our design. Note that these are very short buses spanning four layers. The allocation policy in *3D_sttorus* is exactly the same as *3D_torus*. However, as a consequence of the allocation method, our application generates significant amount of traffic between nodes in the same column, which in turn leads to bus contention and destination contention, as we further show experimentally in Section 4.9.

4.8 Routing and Arbitration

Our routing policy is based on dimension-order: XY routing on folded torus for *2D_serial* and *2D_parallel*; and XYZ routing for *3D_torus* and *3D_sttorus*.

In *2D_serial* and *2D_parallel* systems, we distinguish between messages originating from contiguous partitions and non-contiguous partitions, and the

corresponding flits are designated as *A-type* or *B-type* respectively. Each node has a set of allowed directions depending on the partition it is situated in. For a contiguous partition, any network switch on the partition boundary has channels leading out of the partition marked as *disallowed*. For non-contiguous partitions,

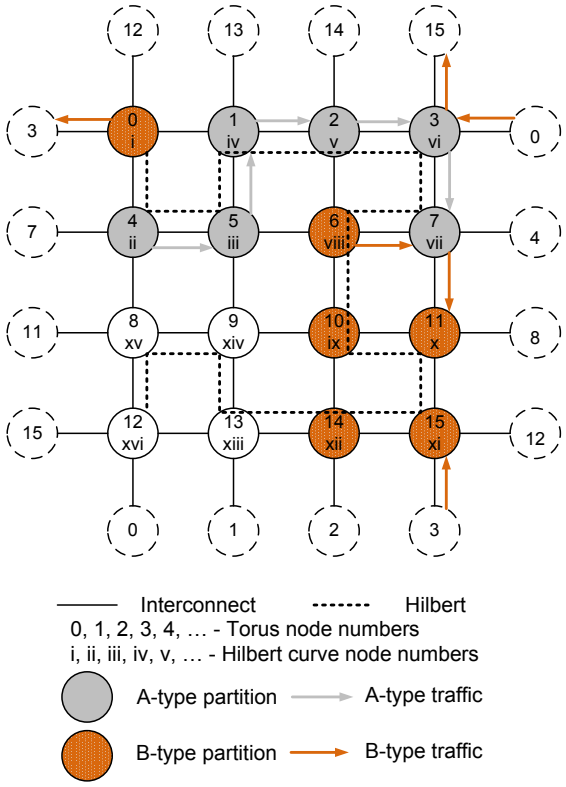


Figure 4-9. Examples of different paths taken while routing A-type and B-type traffic

all network switches have all directions marked as *allowed*. In other words, traffic emanating from a contiguous partition always remains within the partition boundary and traffic emanating from a non-contiguous partition is free to move in any direction dictated by the routing policy. At each network switch, an A-type message is restricted to make the next hop in one of the allowed directions. However, since B-type messages have unrestricted access, they follow torus routing, which is similar to XY routing but includes the torus loopback information to determine the shortest path. A-type messages take the X direction

if that direction is allowed, and Y direction otherwise. Fig. 4.9 shows an example. A message in an A-type partition going from node 4 to node 7 follows the path indicated. In the B-type partition, there is one message going from node 6 to node 11 via node 7 outside the partition. Another message from node 0 to node 15 goes through node 3, which is outside the partition, and makes use of the torus loopback.

It can be noted from the above routing mechanism that switches internal to a contiguous partition will encounter A-type traffic from that partition and may encounter B-type traffic from any non-contiguous partition(s). On the other hand, switches internal to a non-contiguous partition will face only B-type traffic from non-contiguous partition(s). When there is more than one message competing for the same port at any switch, the following arbitration policy is used. The remaining hop-count of the message is determined by looking up the pre-calculated Manhattan distance from the current node to the destination. The message with the maximum remaining hop-count, i.e. the one farthest from its destination, is granted the channel. In case of a tie, B-type is given preference. The remaining hop-count is also used as the arbitration parameter while routing in *3D_torus*. This policy ensures that traffic with a higher potential latency is routed earlier, thereby reducing worst-case latency.

Since B-type messages in *2D_serial* and *2D_parallel* follow XY routing on torus (as described above), any non-contiguous partition is automatically deadlock-free. For contiguous partitions of sizes 2 and 3, there is no possibility of a cycle in the channel dependency graph because the message is always

contained within the partition and 2 or 3 nodes cannot form a cycle on a torus. Hence, deadlock is avoided in this case. For contiguous partitions of size 6, we can have a partition like the A-type partition (nodes 1, 2, 3, 4, 5 and 7) in Fig. 4.9 or a partition comprising of nodes 8, 9, 10, 12, 13 and 14 in Fig. 4.7. In the former case, we do not have a cycle and hence deadlock cannot arise. In the latter case, because we follow XY routing, deadlocks are avoided. For routing in *3D_torus* and *3D_sttorus*, we follow XYZ (dimension-order) routing. Therefore, our routing and arbitration policy for each kind of architecture is deadlock-free.

4.9 Experimental Results

4.9.1 Experimental Setup

The computation core has a datapath width of 64 bits and provides a number representation accuracy of 2^{-52}. We synthesized Verilog RTLs for the computation core, the instruction-decoding wrapper, the routers and *MasterController* with 65 nm standard cell libraries from CMP [88]. The NoC interconnects are laid out and their physical parameters (power dissipation, delay) are determined using the extracted parasitics (resistances and capacitances). Use of folded torus topology prevents occurrence of long warp-back wires. The critical path occurs in the PE datapath as mentioned in Section 4.3.1, following which we used a clock with 1 ns period. We simulated *2D_serial* NoCs with system sizes N=16, and *2D_serial*, *2D_parallel*, *3D_torus* and *3D_sttorus* NoCs with N=64 using the NoC simulator used in [104]. Recall that there are four PEs per NoC node in the system.

The NoC-based multi-core platform is modeled as a co-processor connected using a PCIe interface. We modeled a PCI Express 2.0 interface using Synopsys Designware IP PCI Express 2.0 PHY. This IP has been implemented on 65 nm process and operates at 5.0 Gbps. We use a 32-lane PCIe 2.0 for our simulation.

We ran RAxML-VI-HPC (version 7.0.4) [101] on three inputs that are provided with the suite (Table 5). These inputs comprised of DNA sequences originally derived from a 2,177-taxon 68-gene mammalian dataset described in [108]. We ran RAxML in single and multi-threaded modes on a Pentium IV 3.2 GHz dual-core CPU, and used the best software run-times (four threads or 4T) as our baseline. Furthermore, to measure the relative computation intensities of each

Table 5: Details of test-cases used for running RAxML

Input sequences	A	B	C
Number of sequences	50	50	500
Number of distinct alignment patterns	3066	23385	3829

function kernel, we profiled RAxML on all inputs using the GNU *gprof* utility. The results consistently showed that the functions *coreGTRCAT* (*f3*) (48%), *newviewGTRGAMMA* (*f6*) (21%) and *newviewGTRCAT* (*f2*) (17%) collectively account for more than 85% of the total software run-time, as shown in Fig. 4.10 (a). Fig. 4.10 (b) shows the number of invocations of each of the top three functions. The average CPU times spent in the invocation of each of the three functions were also noted; these times are labeled T_{f2}, T_{f3} and T_{f6}. We generated

100 bootstrap trees using RAxML for each input and used them for subsequent likelihood calculation.

We compared the numerical results produced in our PEs with the ones produced while running RAxML on the above-mentioned CPU using 8 decimal places of precision and verified that the average percentage of deviation was below 0.1%, which was within tolerable limits and did not hamper the stability of RAxML or the likelihood computation.

4.9.2 Test-case Design

The function kernels whose acceleration we target are invoked during generation of bootstrap trees and computation of likelihood of the generated trees to find out the best tree. Working on each tree in parallel helps us work around the sequential dependency among functions within one execution thread.

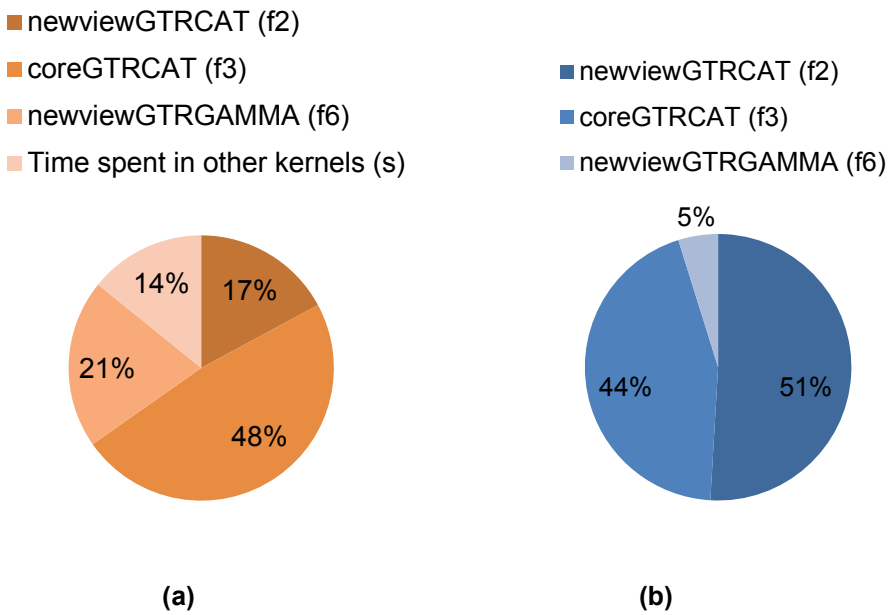


Figure 4-10: Pie charts showing (a) contribution of coreGTRCAT, newviewGTRGAMMA and newviewGTRCAT to the total 1-thread software run-time of RAxML and (b) number of invocations of these functions in typical runs.

Target function kernels originate from different parallel execution threads and hence can be allocated to different nodes of the NoC-based platform. Allocation of nodes to each function is based on the policy in Section 4.7. We designed test cases for *2D_serial* (N=16 and N=64), *2D_parallel* (N=64), *3D_torus* (N=64) and *3D_sttorus* (N=64). Each test case represents a combination of the three target functions (*f2*, *f3*, *f6*). In order to compare the different NoC architectures, we use the same test cases on each. However, the allocation of a test case can result in different mixtures of contiguous and non-contiguous partitions depending on the underlying architecture and system size. Test cases have been captured from a wide range of real world scenarios, including the best and the worst case. The mean execution time of each function is estimated by averaging over all the test case scenarios. During the execution of a function, inter-node exchange of intermediate results occurs simultaneously with intra-node computation (in the PEs within the subnet). This allows masking of communication latency by computation delay. We observed that *newviewGTRCAT* (*f2*) requiring 2 nodes per invocation is generally computation-intensive, while *coreGTRCAT* (*f3*) and *newviewGTRGAMMA* (*f6*) requiring 3 and 6 nodes respectively are generally communication-intensive.

4.9.3 Communication Latency

Total communication latency indicates the amount of time spent in executing the function. Since computation and communication are pipelined, we define *residual communication latency* as the number of clock cycles spent in performing only inter-node communication. The average lifetime of each partition is closely related to the total communication latency.

The contiguity (or non-contiguity) of an allocated partition has a direct bearing on the communication latency (total or residual) for executing the function. The effect is most pronounced in the case of *newviewGTRGAMMA* (*f6*) and also affects *coreGTRCAT* (*f3*). On the other hand, *newviewGTRCAT* (*f2*) has a net zero residual communication latency. This is because this function is spread across only two nodes and computation and communication cycles complement each other. We use average partition dispersion (diameter) as a measure of the non-contiguity of the allocated partition. We observe (Fig. 4.11) a gradual decline in average partition dispersion moving from *2D_serial* to *2D_parallel* to *3D_torus*. The average communication latency involved in

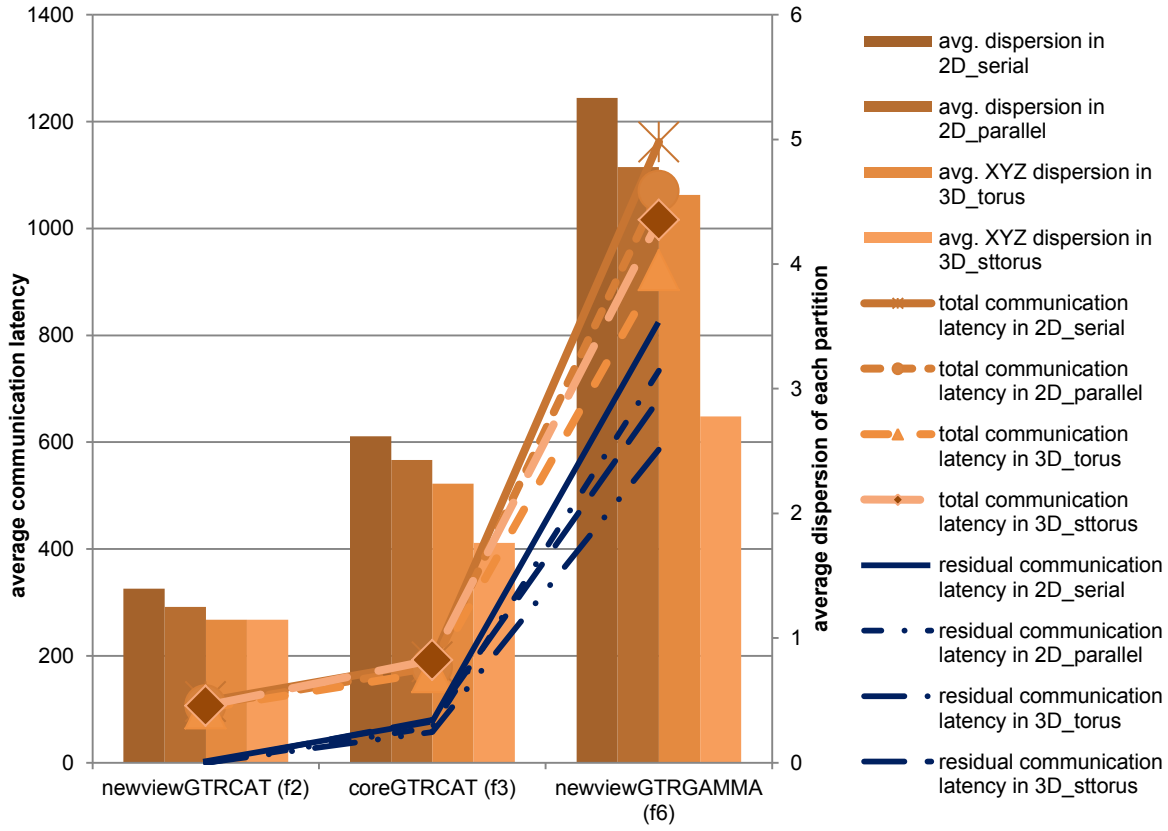


Figure 4-11: Variation of partition dispersion and function communication latency across different NoC architectures

function execution displays a similar trend across architectures. The role of the interconnection topology here is to reduce the average partition dispersion and hence the residual communication latency. B-type messages originating from non-contiguous partitions as a percentage of the total number of messages reduce from 34% in *2D_serial* to 24% in *2D_parallel*, further demonstrating the impact of contiguity of partitions on latency performance. Partitions on *3D_sttorus* have much lower dispersion than the other architectures owing to the presence of a bus in the vertical dimension, which provides one-hop transit between any two layers. However, this does not translate to lower communication latency because of bus and destination contention. In fact, latencies for *3D_sttorus* are observed to be slightly higher than those in *3D_torus* (most pronounced for *f6* as shown in Fig. 4.11).

4.9.4 Speedup

We used two different measures to evaluate the acceleration performance of our design. The first measure is function-level speedup, which assesses the level of fine-grained parallelism achieved by our PE design. The second measure is aggregate speedup of the accelerated kernels, which measures the degree of acceleration achieved by integrating the PEs in the NoC framework.

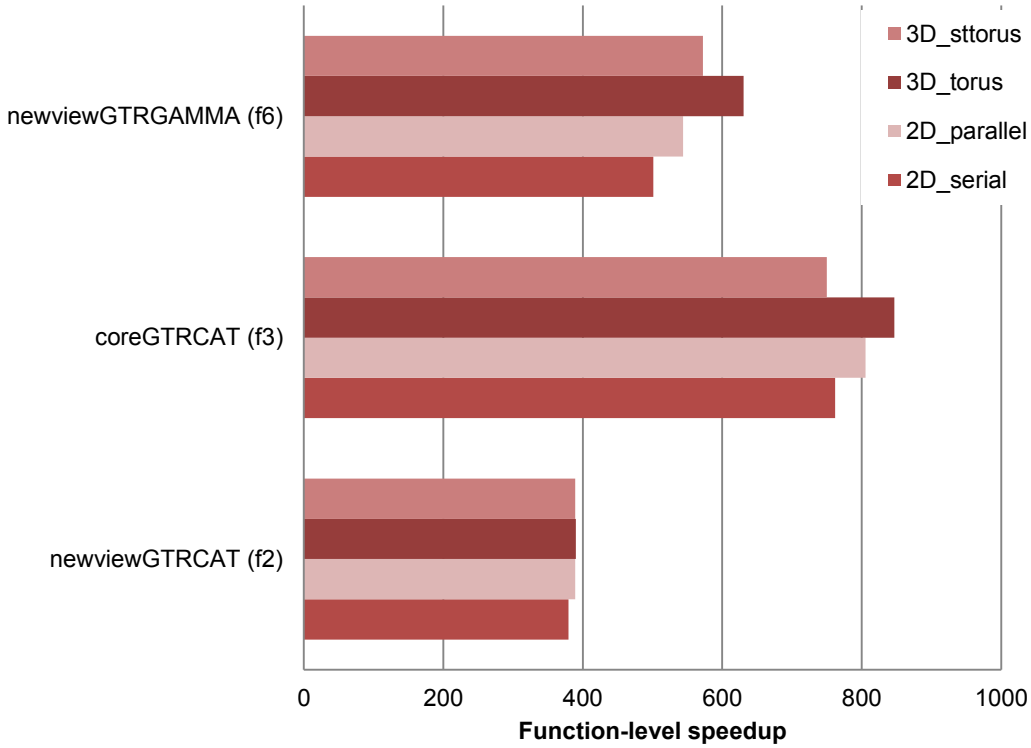


Figure 4-12: Function-level speedup of phylogenetic kernels

4.9.4.1 Function-level Speedup

In order to determine function-level speedup, the total execution time for each function, consisting of computation and communication components, was averaged over all test cases on each architecture (*2D_serial*, *2D_parallel*, *3D_torus* and *3D_sttorus*) and compared with the baseline CPU times consumed by the function while running the software (T_{f2}, T_{f3}, T_{f6}). The speedup obtained for the functions on each architecture is shown in Fig. 4.12. *3D_torus* consistently provides the best function-level speedup for all three functions. Note that the best speedup (on *3D_torus*) of 847x is obtained for *coreGTRCAT* (*f3*), which accounts for 48% of the total software run-time. The least speedup (on *3D_torus*) of 390x is obtained for *newviewGTRCAT* (*f2*), because it is the smallest function kernel and requires only two NoC nodes (or 8 PEs) by design.

As expected, function-level speedup has an inverse relationship with communication latency (Fig. 4.11).

4.9.4.2 Aggregate Speedup of the Target Function Kernels

This is a measure of the acceleration achieved on the targeted function kernels, and is the ratio of the CPU run-times of the test cases consisting of these kernels to the run-times of these test cases on our NoC-based platform of a given system size (*N*) and architecture (*2D_serial*, *2D_parallel*, *3D_torus* and *3D_sttorus*). Each test-case configuration represents a typical snapshot of the system during the course of execution of parallel RAxML threads, with our NoC-

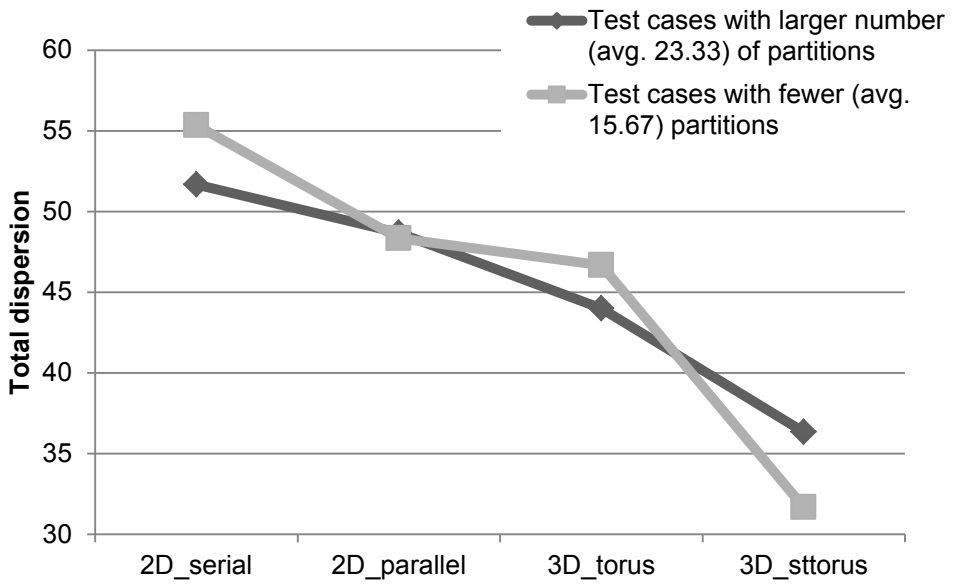


Figure 4-13: Total dispersion of target kernel mappings across different NoC architectures

based platform handling the three phylogenetic kernels. Several instances of *newviewGTRGAMMA* (*f6*), *coreGTRCAT* (*f3*) and *newviewGTRCAT* (*f2*) occupying contiguous and non-contiguous partitions are present in each such test-case. The total time spent in one test case also includes the time required to allocate all partitions (*allocation time*) and to load the input vectors to the

function in 64-bit FXP-HNS format [83] (*interface time*) on the NoC using the PCIe interface described earlier.

On an average, *2D_serial* with N=16 provides a speedup of ~2200x, whereas a larger system size (N=64) provides ~4300x speedup. The ideal increase (4x) in speedup with system size was not obtained because of higher penalties incurred in *allocation time* and *interface time*, and higher non-contiguity of partitions leading to increased communication latency. This is where the benefits provided by *2D_parallel*, *3D_torus* and *3D_sttorus* become

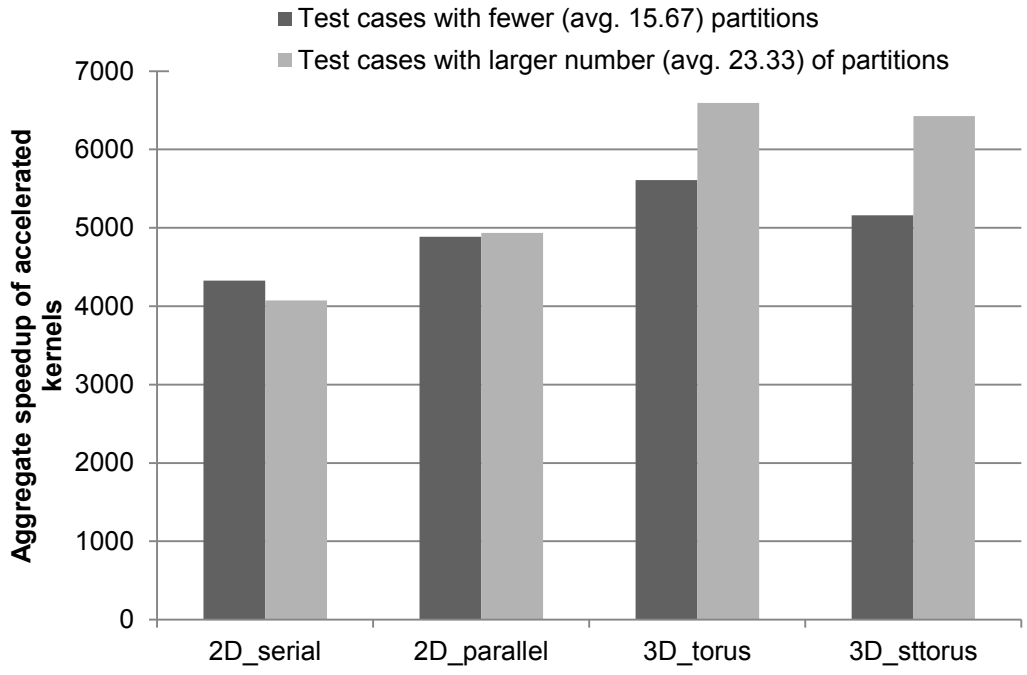


Figure 4-14. Average aggregate speedup of the accelerated kernels across different NoC architectures

evident.

We classified test cases on systems with N=64 on the basis of the number of constituent functions (or partitions). Test cases with a lower number of partitions (average 15.67) have more instances of *f6*. Such instances occur

mainly during the likelihood evaluation phase. Test cases with higher number of partitions (average 23.33) have significantly more instances of *f2* and *f3*. These scenarios are prominent during generation of bootstrap trees. Fig. 4.13 shows the observed dispersion as a function of the underlying architecture and the number of partitions. A test case with fewer partitions is expected to result in a higher degree of dispersion because there are more instances of *f6*. This is generally true for all the architectures except for *3D_sttorus* because of the bus. Fig. 4.14 shows the aggregate speedups of the accelerated kernels as a function of the underlying architecture and the number of partitions. *2D_parallel*, *3D_torus* and *3D_sttorus* NoCs provide higher speedup than *2D_serial* because they reduce *allocation time* while improving partition contiguity. For all test cases, *3D_torus* provides the best aggregate speedup (6594x) followed by *3D_sttorus* (6428x), *2D_parallel* (4937x) and *2D_serial* (4326x). *3D_torus* outperforms *3D_sttorus* because bus and destination contention in *3D_sttorus* leads to higher communication latency (as described earlier in Section 4.7.4.3).

4.9.5 Total Execution Time

In order to determine the overall reduction in run-time, the run-time of the non-accelerated portion of the software is considered along with the accelerated portion running on the NoC-based platform. The total execution time takes into account all overheads involved in offloading a part of the computation to the NoC-based platform. Table 1 shows the total run-times for two representative input data-sets, 50_5000 containing 50 DNA sequences with 5000 columns each and 500_5000 containing 500 DNA sequences with 5000 columns each. Table 6 shows the total run-time using our *2D_serial*, *2D_parallel*, *3D_torus* and

Table 6. Total run-times for different inputs using different NoC-based platforms vis-à-vis only software

Input data (DNA)		Unaccelerated software run-time (s)	Time spent in accelerated kernels (s)	Allocation time (s)	PCIe interface time (s)	Total run-time using NoC platform as hardware accelerator (s)	Total 4T software run-time (s)
50_5000	2D_serial	292.000444	0.515478	0.130387	0.145065	292.791374	924.052039
	2D_parallel	292.000444	0.481303	0.104805	0.145065	292.731617	924.052039
	3D_torus	292.000444	0.433625	0.050889	0.145065	292.630024	924.052039
	3D_sttorus	292.000444	0.474657	0.050889	0.145065	292.671056	924.052039
500_5000	2D_serial	7038.847538	19.1142	8.467062	8.273363	7074.702162	37124.7233
	2D_parallel	7038.847538	18.04733	6.805803	8.273363	7071.974034	37124.7233
	3D_torus	7038.847538	16.766102	3.304655	8.273363	7067.191658	37124.7233
	3D_sttorus	7038.847538	18.102936	3.304655	8.273363	7068.528491	37124.7233

3D_sttorus architectures vis-à-vis software. The best run-time reduction is obtained using *3D_torus* NoC-based platform and is highlighted in the table. It can be observed that most of the run-time that results from the use of the NoC-based platform comes from the unaccelerated portion. Even so, the overall run-time is reduced by more than 3x for 50_5000 and more than 5x for 500_5000. This proves the immense potential of such hardware accelerator platforms in the field of phylogeny reconstruction applications.

4.9.6 Energy consumption

Fig. 4.15 shows the total energy consumed across different test cases and architectures. Test cases with larger number of partitions consume more energy than those with fewer partitions on the same architecture. However, there is a significant reduction (up to 37.7%) of energy going from *2D_serial* to *3D_torus*. This follows a trend similar to that observed for total test case dispersion (Fig. 4.13). Lower dispersion leads to lower average hop-count of inter-node messages and hence lower energy consumed in communication. Also, in the case of 3D (both *3D_torus* and *3D_sttorus*), substitution of longer horizontal links with much shorter vertical links leads to lower energy consumption. *3D_sttorus* has a

slightly higher overall energy consumption over *3D_torus* because of the higher capacitance of the buses and higher application run-times.

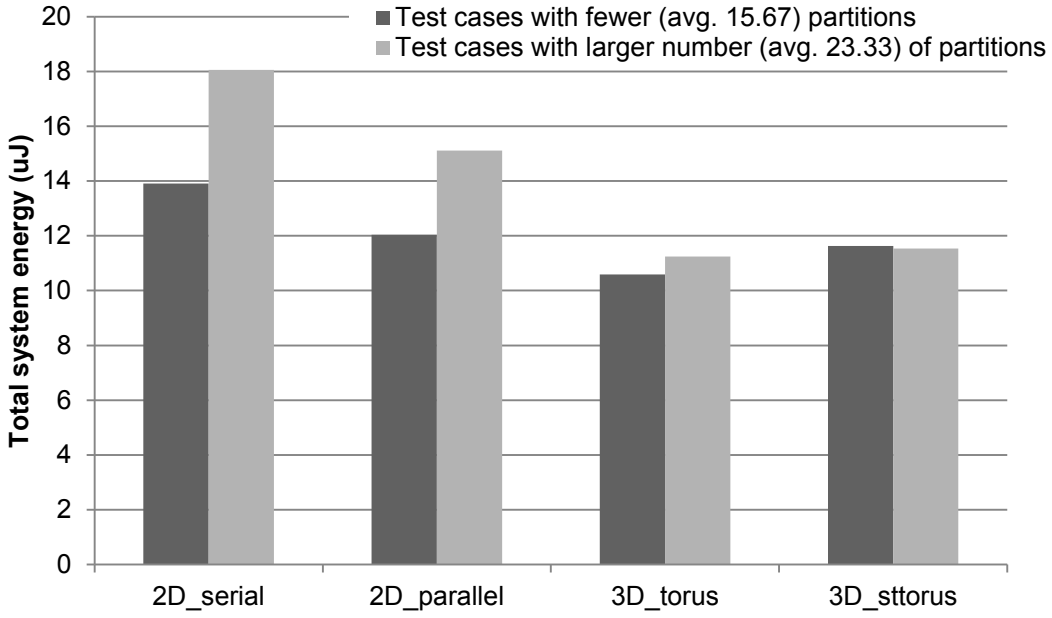


Figure 4-15. Total system energy consumption across different NoC architectures

4.10 Conclusion

In this chapter, we presented a novel design and implementation of a Network-on-Chip (NoC) based multi-core platform for accelerating Maximum Likelihood (ML) based phylogeny reconstruction, which is an important, compute-intensive application in bioinformatics. The NoC-based accelerator targets the three most time-consuming function kernels that collectively account for the bulk of the run-time in the widely-used RAxML software suite. Our implementation achieves parallelization at different levels – both within a function kernel and across several invocations of these function kernels in parallel execution threads. Consequently, our contributions include: (i) the design of a fine-grained parallel PE architecture, (ii) a novel algorithm to

dynamically allocate nodes to tasks based on Hilbert space-filling curves, and (iii) the design and extensive evaluation of different NoC architectures, both in 2D and 3D, in the context of this application. The overarching purpose of our experimental study was to evaluate the feasibility and merits of an NoC-based hardware accelerator for ML-based phylogenetic kernels. To this end, our experimental results show that our NoC based accelerators are capable of achieving a function-level speedup of ~847x, aggregate speedup of the accelerated portion up to ~6,500x, and overall run-time reduction of more than 5x over multithreaded software. Comparative evaluation across NoC architectures show that the best performances in terms of speedup and energy consumption are obtained from 3D NoC platforms. Our speedup performance represents considerable improvement over existing hardware accelerators for this application (e.g. [69], [100]).

Although this work targeted the RAxML implementation of ML phylogeny, the design methodology and ideas for node allocation and routing are generic enough to be carried forward to other scientific applications which have a similar computational footprint, i.e., the need to execute a large volume of a fixed number of function kernels; for example, other statistical estimation methods in phylogenetic inference such as Bayesian Inference.

5. High-Throughput, Energy-Efficient NoC-Based Hardware Accelerators

In all fields of high-performance computing (HPC), new data-generation technologies are placing an enormous stress on software tools to perform beyond terascale to peta- and exa-scale. Given the diversity of tools and the need to cater to a wide user-base, it is becoming common practice, even within academic settings, to have a dedicated center that hosts a variety of scientific computing tools on a few high-end data servers. The throughput requirement that these multi-user servers need to meet can be substantial since the servers can be expected to service concurrent requests from a variety of applications, each with differing resource requirements. These servers would often consist of multicore hardware accelerator co-processor platforms where the cores are designed to accelerate targeted operations and are interconnected with an on-chip network. A similar setup is also becoming common practice, albeit on a larger scale, in cloud solution providers (e.g. [109]). While throughput is important, it is necessary to restrain energy consumption and power dissipation in these hardware accelerators. In this respect, the role of NoC-based platforms assumes significance, owing to the level of on-chip integration that such platforms can achieve, delivering high computation throughput alongside energy-efficiency, as we elaborate in this chapter.

In this chapter, we propose, design and evaluate NoC-based platforms for enhancing the computation throughput of scientific applications. Our evaluation of the NoC-based platforms includes analysis of the resulting performance, energy-efficiency and power consumption, and thermal profiling. More

specifically, we analyze how throughput and power consumption are correlated with the statistical properties of the application traffic. In addition, we compare and analyze chip-level thermal profiles to identify hot-spot distribution and correlate them with architecture-level design choices.

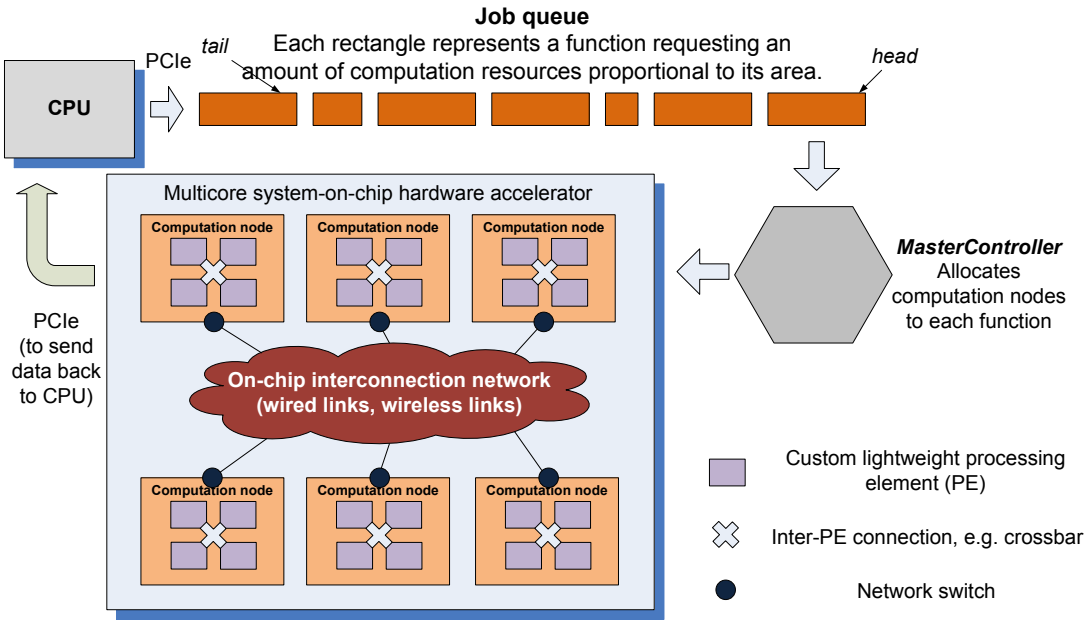


Figure 5-1. Illustration of our NoC-based use-case model proposed for hardware acceleration

5.1 Application Use-Case Model

In order to design our platform, we propose the following *use-case model* (see Fig. 5.1): A CPU runs the parent process and communicates via an interface (e.g. PCIe) to a multicore system-on-chip that acts as a hardware accelerator for specific computation-heavy kernels. There is a queue of jobs offloaded by the CPU to the hardware accelerator and an allocation unit (*MasterController*) assigns the requested computational resources from the hardware accelerator to the job at the head of the queue. Once some computation resources are assigned to a job, they stay busy till the execution of that particular job concludes, and the result is sent back to the CPU through a similar interface (e.g. PCIe). Each

computational resource is a lightweight custom core embedded in a NoC. In the following, we characterize such hardware accelerator platforms in terms of overall computation throughput, energy consumption, power dissipation and thermal profiling.

The computational footprint of many scientific applications fits the proposed use-case model, for example, the use of servers that host application programs to implement standard functions in phylogenetic inference [110], genome/gene sequencing [89], climate modeling and weather prediction [111], etc. For instance, a typical genome assembly algorithm farms out billions of pairwise sequence alignment tasks, each of which aligns two strings of small lengths (e.g., 100-500 base pairs) and can use a small number of cores (e.g., 8-16) [112]. As another example, consider the problem of computing phylogenetic inference using maximum likelihood (ML) [113], where one typically needs to carry out billions of independent tree evaluations, each of which internally performs a small number of floating point calculations using a few cores. In such applications, enhancing overall throughput in computation translates to shorter time to solution. To this end, integration of many cores using an on-chip network (or NoC) presents an attractive model of computation, not only due to the availability of a large number of cores, but also because the computation within the individual tasks in many of these applications (e.g., sequence alignment in the genome assembly problem, [114]) can be designed to take advantage of fine-grain on-chip parallelism involving a fixed number of cores.

5.2 Introduction of Long-Range Links

For any NoC design, the choice of the on-chip network architecture is an important determinant of the overall throughput and energy efficiency achieved by the many-core system.

Introduction of long-range links on regular structures such as a mesh leads to an interconnect fabric that is neither regular nor fully customized, as shown in [115]. This has the effect of reducing average packet latency and increasing data throughput without any major impact over the network topology. Most complex networks, including the Internet, social networks and the network of neurons in the brain share this Small-World property [116]. In [115], the use of on-chip wireless interconnects to implement these long-range links is demonstrated to provide considerable gains in network throughput and latency, and energy consumption over wired counterparts.

Consequently, the focus of this chapter is the design and evaluation of NoC-based platforms for throughput-oriented applications. These platforms consist of long-range on-chip wireless links in addition to standard wired links in a particular network topology. To the best of our knowledge, this represents the first attempt at designing and empirically characterizing NoC-driven accelerator platforms built using both wired and wireless links for throughput-oriented scientific applications.

5.3 Network Architecture

The choice of the underlying interconnection fabric topology is determined from the perspective of the application and VLSI implementation. Our target is

the class of applications that spawn a stream of independent jobs (constituent functions) that individually require variable amounts of computation resources. Communication occurs only among nodes catering to a single job during its execution. The location of these nodes on the network is a variable for every instance of an allocated job, leading to arbitrary point-to-point communication. The traffic patterns are hence dynamically changing and steady-state characteristics do not indicate any clustering or traffic localization, as shown in a sample statistical analysis of the traffic pattern over time in Fig. 5.2.

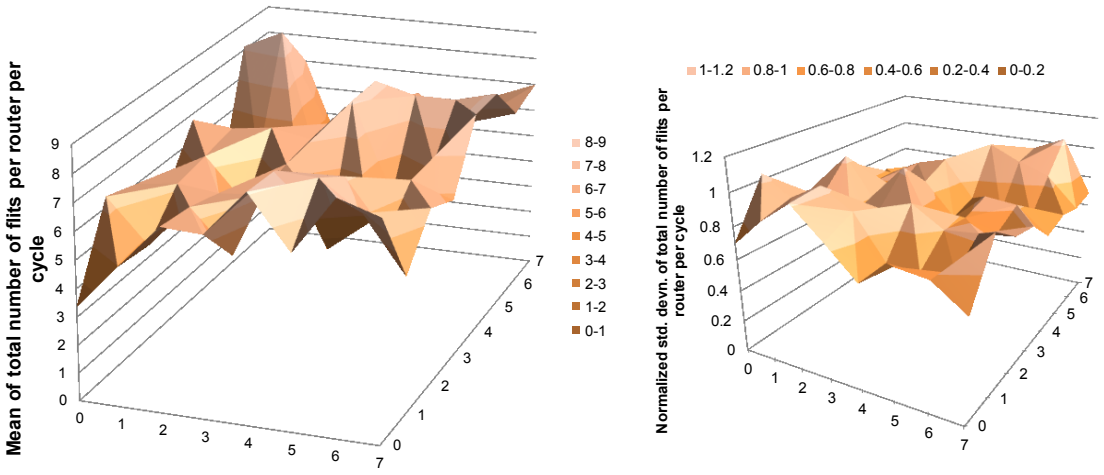


Figure 5-2. The number of flits per cycle ((a) mean and (b) normalized standard deviation) within routers of an 8x8 folded torus network. The horizontal dimensions denote the processor coordinates of the torus, while the vertical dimension denotes the observed flits per cycle (mean and standard deviation). The absence of distinctive peaks in temporal statistics indicates the higher suitability of fully-distributed, regular interconnection topologies such as a mesh or torus, as opposed to linear or hierarchical topologies such as bus or trees.

Distributed network architectures are generally better suited for such traffic patterns. Consequently, we use a *folded torus*. From the VLSI implementation perspective, a torus is a scalable network architecture whose regularity provides for easier timing closure and reduces dependence on interconnect scalability [27].

We adopt the computation nodes described in Chapter 4, Section 4.3, which run at a clock speed of 1 GHz. All inter-node links in the folded torus are one-hop links with respect to the 1 GHz clock used. Since the computation node has a datapath that is 64-bit wide, we designate our flit size to be 64 bits and split each inter-node message into three flits – header, body and tail. As a result, each inter-node link has a minimum bandwidth of 64 Gbps.

5.4 Use of Long-Range On-Chip Wireless Links

In the previous section, we described our underlying network topology that suits a distributed traffic pattern. However, we would ideally want the average internode distances between nodes catering to the same job to be as low as possible, or in other words, we would want the nodes catering to a particular job to be all contiguous to one another. This cannot be guaranteed in practice because different jobs needing different number of resources could get submitted in real-time (as we further explain in Section 5.5, also see Section 4.7). This could force any allocation method to either wait for all required nodes to be available contiguously (the effect of which could be a significant delay in execution time coupled with a non-optimal use of the cores) or map the job on nodes that could potentially be non-contiguous along the network (as elaborated in Section 5.5). In the latter approach, large physical separation of these nodes on the network could lead to a significant communication overhead. From the network architecture point of view, bridging these gaps is possible through the use of long-range point-to-point shortcuts. In Section 3.2, we described our underlying network topology that suits a distributed traffic pattern. However, we would ideally want the average internode distances between nodes catering to the same

job to be as low as possible, or in other words, we would want the nodes catering to a particular job to be all contiguous to one another. This cannot be guaranteed in practice because different jobs needing different number of resources could get submitted in real-time (as we further explain in Section 3.4). This could force any allocation method to either wait for all required nodes to be available contiguously (the effect of which could be a significant delay in execution time coupled with a non-optimal use of the cores) or map the job on nodes that could potentially be non-contiguous along the network (as elaborated in Section 3.4). In the latter approach, large physical separation of these nodes on the network could lead to a significant communication overhead. From the network architecture point of view, bridging these gaps is possible through the use of long-range point-to-point shortcuts. As mentioned before, introduction of shortcuts on regular architectures have been shown to provide significant improvements in latency and network throughput for different kinds of applications [115]. Implementing these shortcuts using metal wires inherits the issues associated with long wires, viz., transmission delay and large power dissipation. High transmission delay makes it impossible to guarantee one-hop transmission. Use of on-chip wireless shortcuts overcomes these drawbacks [117].

5.4.1 Physical Layer

Suitable on-chip antennas are necessary to establish the wireless links. It has been shown that wireless NoCs designed using carbon nanotube (CNT) antennas can significantly outperform conventional wireline counterparts [117]. Antenna characteristics of CNTs in the THz frequency range have been

investigated both theoretically and experimentally [118]. Such nanotube antennas are good candidates for establishing on-chip wireless communications links and are henceforth considered in this work. CNT antennas can be used to assign different frequency channels to pairs of communicating source and destination nodes. This enables creation of dedicated and non-overlapping channels using the concept of frequency division multiplexing. This is implemented by using CNTs of different lengths, which are multiples of the wavelengths corresponding to the respective carrier frequencies. With currently available technology, it is possible to create 24 non-overlapping wireless channels, each capable of sustaining a data rate of 10 Gbps using CNT antennas. Technology-specific details on CNT are discussed in [117]. We determine the number of wireless links in our system based on the bandwidth each link needs to support. As mentioned earlier, each wireless (inter-node) link needs to sustain a bandwidth of 64 Gbps. Since each wireless channel can provide a bandwidth of 10 Gbps, we need to combine 7 channels per link (delivering up to 70 Gbps bandwidth). Hence, the maximum number of single-hop wireless links we can implement is $\lfloor 24/7 \rfloor = 3$. Note that we could increase the number of wireless links providing the same bandwidth when future technology supports more than 24 non-overlapping channels.

5.4.2 Wireless Link Placement

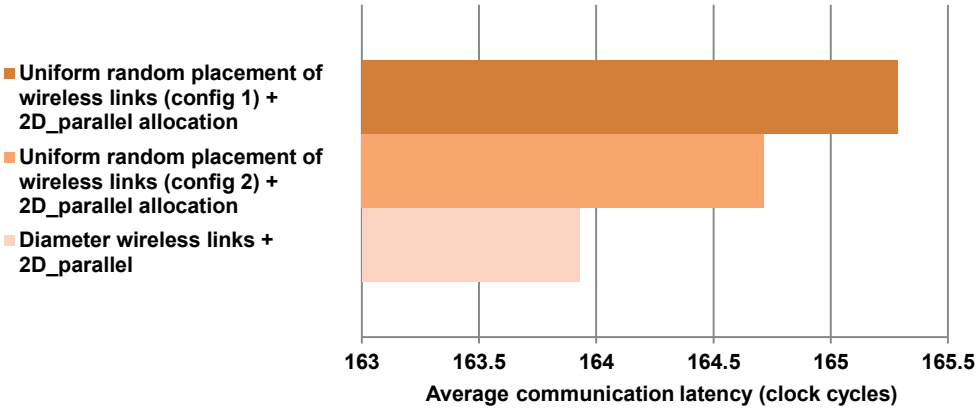


Figure 5-3. Comparison of average network communication latencies for different wireless link placements. Even small increases in communication latency have shown to lead to significantly degrade performance.

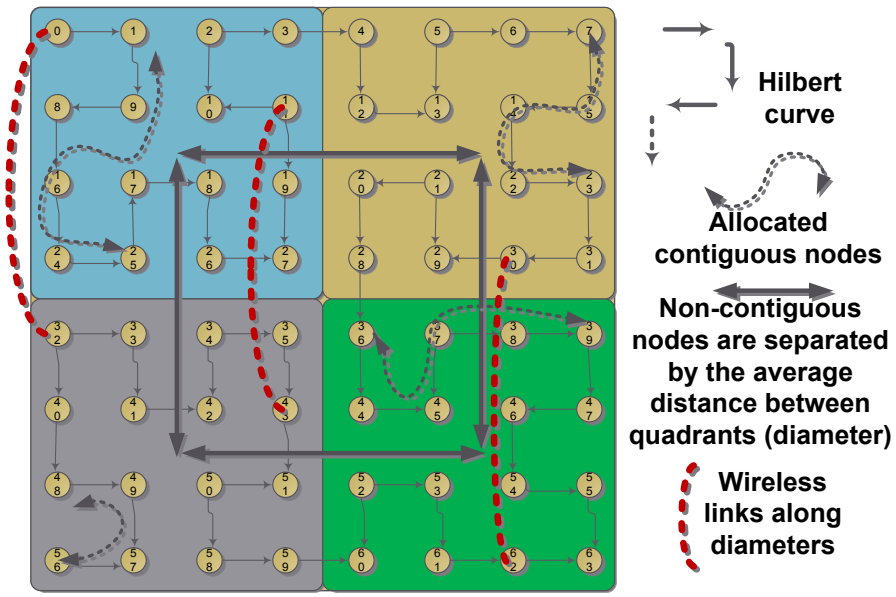


Figure 5-4. Non-contiguous nodes and long-range communication requirements leading to wireless link placement along diameters. Although not shown, each node is connected through wired links to four of its neighboring nodes as dictated by the torus topology. Most of our allocation strategies consider the nodes along the Hilbert curve while also factoring the presence/absence of wireless hubs at intermediate nodes..

Ideally, the placement of wireless links should be dictated by the demands in the traffic patterns generated by the target scientific application. For the kind of throughput-oriented scientific applications targeted in this paper, however, it is not possible to statically predict any particular traffic pattern because the

underlying communication requirement could be arbitrary. The sets of communicating nodes executing a single task could be spread out over the whole network, thereby generating arbitrary point-to-point traffic over time (as corroborated by our observations made in Fig 5.2). In fact, we observed the probability of non-local interaction between nodes is highest when the separation between them is equal to the diameter of the network. We have experimentally verified this observation by placing wireless links according to various considerations – based on uniform random traffic assumption, and along diameters of the network. As shown in Fig. 5.3, wireless link placement along the diameter leads to the lowest network latency among the possibilities considered. This observation can be explained by the fact that the most efficient node allocation method described later in Section 5.5 divides the network into four quadrants, tries to allocate nodes locally, and the need for long-range links arises when allocated nodes are non-contiguous and lie in neighboring quadrants. It can be easily seen from Fig. 5.4 that the mean distance between adjacent quadrants is the network diameter of the folded torus. Since there are only 3 wireless links as explained earlier, we maximize their coverage by placing them along diameters of the folded torus with almost equal angular separation between each pair of diameters, as shown in Fig. 5.4. This ensures that nodes in all sections of the network have similar degree of accessibility to a wireless link.

5.5 Dynamic Node Allocation

A network node is *busy* during the execution of a job by the PE; it is *available* otherwise. The computation nodes (PEs) continually send their busy/available status to the allocation unit, *MasterController* (see Fig. 5.1).

When a job requests computation resources, *MasterController* allocates the requisite number of available computation nodes from the system. The nodes thus allocated form a *partition* during the course of function execution and communicate with one another (also see Section 4.7). A desired feature of a partition is that its constituent nodes are co-located so as to minimize the average number of hops spent in message transfers. To this end, a good allocation strategy should ensure co-locality without incurring a large allocation time overhead. Simple approaches like breadth-first search do not fit these criteria. We present the following allocation methods, which can be classified into *wireless-agnostic* and *wireless-aware* methods. We also make use of the locality-preserving, space-filling Hilbert curve [106] for allocation. The resultant allocated partitions are denoted *A-type* if all nodes belonging to that partition are contiguous along wired links on the folded torus; else the partition is *B-type*.

5.5.1 Parallel Best-Fit Allocation Using Multiple Hilbert Curves

This allocation strategy preferentially looks for a partition with contiguous nodes to maximize co-locality, and parallelizes the search in order to increase the probability of a hit. The algorithm follows the one in Section 4.7.2, and is reproduced here for clarity.

1. First, we use four Hilbert curves on a square folded torus. These four curves are obtained by using right-angle rotation operations of a single Hilbert curve.
2. We further divide each of the four Hilbert curves into four *segments*, one from each quadrant – thereby resulting in a total of 16 segments (see Fig.

5.4). *MasterController* now has 16 heads, each of which is responsible for scanning a segment. All 16 heads act in parallel.

3. Each head now preferentially looks for an A-type partition in its segment. The first head to find such a partition returns it to the requesting job and interrupts all the other scanning heads.
4. In case no A-type partition is found after each head has finished scanning its segment, *MasterController* carries out a serial scan along a Hilbert curve and allocates available nodes as they are encountered.

This method of allocation is wireless-agnostic because we do not make use of the information regarding the location of wireless shortcuts. Systems using this method of allocation are denoted by *2D_parallel* if they do not use wireless shortcuts, and *2D_parallel + wireless* if wireless shortcuts are used only during message transfers.

5.5.2 Wireless-First Allocation Using Hilbert Curve

This is a wireless-aware allocation method in which *MasterController* looks for available node pairs directly connected by a wireless shortcut. If such a pair is available, they are allocated to the requesting job. *MasterController* then serially scans for the remaining nodes following a Hilbert curve starting from a terminal node of the wireless shortcut. Since only nodes belonging to the same partition communicate with one another, this method ensures that wireless shortcuts are fully utilized. In case no wireless shortcut is available at the time of allocation, nodes are allocated based on a serial scan along the Hilbert curve. Systems using this allocation method are denoted by *wireless + Hilbert*.

5.5.3 Wireless-First, Column-Major Allocation

This is another wireless-aware allocation method, which looks for available wireless shortcuts to be allocated first. The remaining nodes are allocated following the direction of wireless shortcuts such that the nodes in the partition are aligned with the shortcut, so as to maximize the traffic the shortcut carries. As shown in Fig. 5.4, the wireless shortcuts are placed along the vertical diameters (columns) of the folded torus. Hence, the node allocation also follows a *column-major* ordering. The major benefit of this method is that a wireless shortcut can potentially carry traffic from partitions that do not directly include it but are closely aligned with it. Systems using this allocation method are denoted by *wireless + column-major*.

5.6 On-Chip Routing

We adopt wormhole routing to exchange three-flit messages among nodes of a partition. Network switches are based on the designs presented in [104]. Each switch consists of four bidirectional ports (E, W, N, S) to neighboring switches and one local port to/from the computational node. Each port has a buffer depth of two flits and each physical channel is split into 4 virtual channels. The general routing policy is e-cube routing on torus [52].

For routing in the presence of wireless shortcuts, we need information about the wireless links closest to a source-destination pair, and the bandwidth provided by such links. This information is known beforehand and is available to the router. Based on this knowledge, the router chooses a path via a wireless shortcut if that entails fewer hops to transfer a message between a source-

destination pair. The message follows deadlock-free south-last routing [115] when involving wireless shortcuts, and e-cube routing when following wired-only paths between a source and a destination.

5.7 Experimental Results

5.7.1 Experimental Setup

The computation core (originally from Section 4.3) has a datapath width of 64 bits and provides a number representation accuracy of ~10^{-15}. A PE integrates four such computation cores. We synthesized Verilog RTLs for the PEs, the network switches and *MasterController* with 65 nm standard cell libraries from CMP [88]. Our clock period of 1 ns comes from the critical path constraint in the core datapath as mentioned in Section 4.3.1 and shown in Fig. 4.1. We verified that our design meets all timing constraints, and evaluated power consumption. We laid out the wired NoC interconnects and determined their physical parameters (power dissipation, delay) using the extracted parasitics (resistances and capacitances). We verified that all wired links could be traversed within one clock cycle.

Each wireless link consists of seven channels of 10 Gbps each, providing a total link bandwidth of 70 Gbps. For the wireless links, we considered an energy dissipation of 0.33 pJ/bit as reported in [117] to include the energy consumed in the transceiver circuitry and the antennas, and used these to evaluate the total energy consumption of our system. In order to carry out chip-level thermal analysis we used the data on power consumption so obtained with HotSpot 5.0,

an accurate and fast thermal model suitable for use in architectural studies [122].

We experimented with the allocation methods mentioned in Section 5.5. We used system sizes of N=64 and N=256 in our experiments. We modeled the NoC-based multicore platform as a co-processor connected using a PCIe interface. We modeled a PCI Express 2.0 interface using Synopsys ™ Designware ™ IP PCI Express 2.0 PHY implemented on 65 nm process and operating at 5.0 Gbps. We used a 32-lane PCIe 2.0 for our simulation.

For experimental studies, we use function kernels from a Maximum Likelihood-based phylogenetic reconstruction software called RAxML version 7.0.4, [70], [101]. A detailed profiling of RAxML runs using the GNU *gprof* utility reveals that a small set of functions consume a predominant portion (>85%) of the runtime. These functions are offloaded to our NoC-based accelerator co-processor and are denoted by *f6*, *f3* and *f2* respectively based on the computation resources (number of computation nodes) they need for execution. Based on the composition of jobs executing on our system, we bin the system job loads into two categories – one in which *f6* jobs are dominant and the other in which *f3* and *f2* jobs occupy up to half of all the nodes. The total number of jobs concurrently executing on the system is clearly higher in the latter case. Since each *f6* individually requires the largest number of computation nodes (six), the probability that one will be allocated a contiguous partition on the network is relatively low. Therefore, the above test plan represents the conservative end of the spectrum for performance evaluation.

5.7.2 Computation Throughput

To measure the computation throughput of our system, we only use each basic operation (logarithm/exponentiation) performed by a core (see Section 4.3.1) as the unit (leaving out addition because it is much simpler), and the number of operations per second as the metric. Computation throughput is not

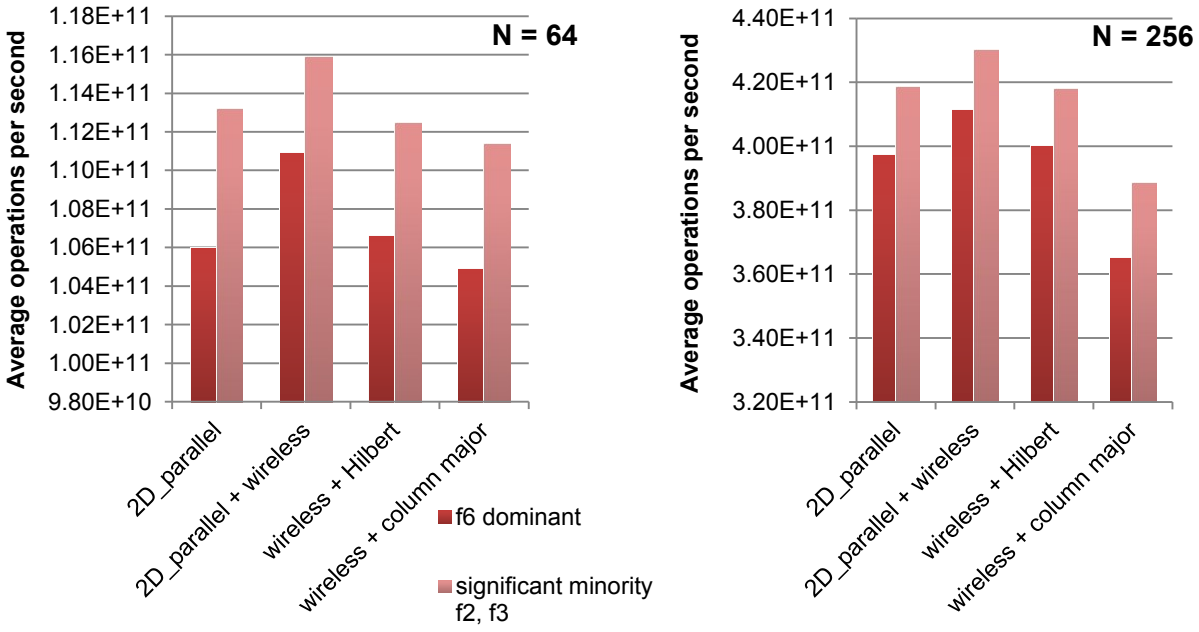


Figure 5-5. Computation throughput across different network architectures, system sizes and job loads.

only affected by the mix of jobs running on the system at any point of time, but also by allocation time overhead, usage of wireless shortcuts, and network architecture. Fig. 5.5 shows the computation throughput for the two different job loads mentioned earlier across different network architectures and system sizes. *2D_parallel + wireless* consistently provides the best computation throughput across job loads and system sizes. It is interesting to note that the best performing architecture has a wireless-agnostic allocation method. While wireless-aware allocation methods guarantee that a larger proportion of flits use the wireless shortcuts (see Fig. 5.6), this also leads to congestion over these links.

Since we use a static routing technique that is based only on the comparison of distances traversed in alternative paths (using shortcuts vs. not using shortcuts), we end up routing more flits through the wireless shortcuts than their bandwidth can sustain without incurring a latency penalty. In a wireless-agnostic allocation method such as *2D_parallel + wireless*, we try to maximize the number of A-type partitions during allocation, leaving to the wireless shortcuts the job of carrying traffic from B-type partitions.

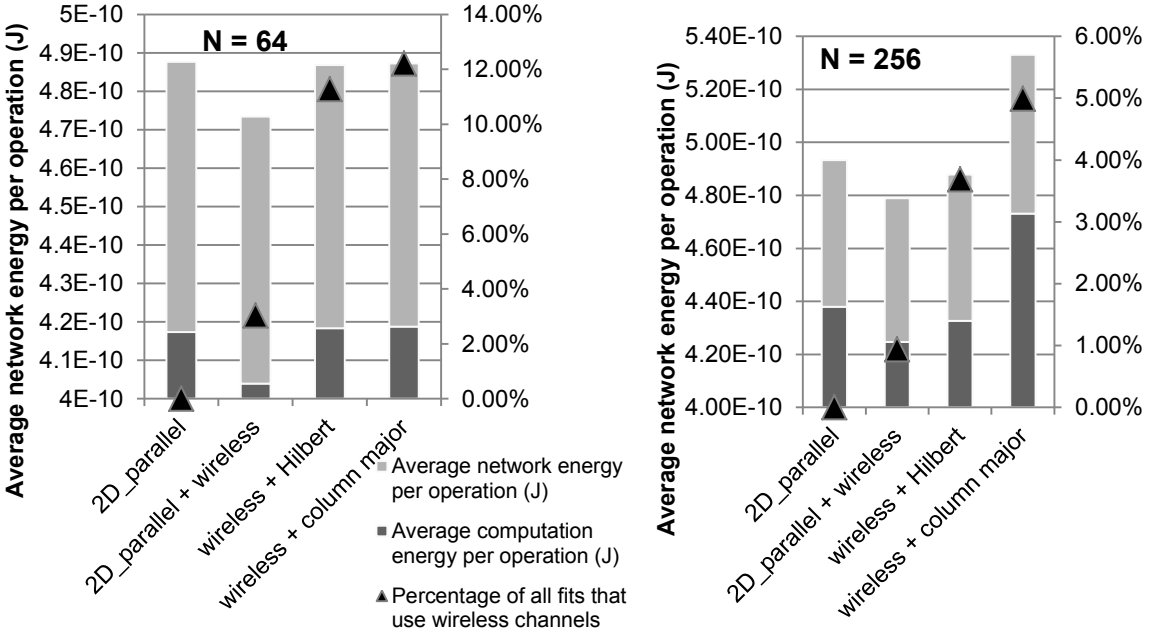


Figure 5-6. Proportion of flits using wireless shortcuts and energy consumption across network architectures and system sizes.

Referring to Fig. 5.5, we also note that the cases containing a higher proportion of *f2* and *f3* jobs have a 5-10% higher computation throughput than the *f6* dominant loading scenario. Note that a larger system size (Fig. 5.5 (b)) provides proportional gain in computation throughput because the problem size can be appropriately scaled up. The lowest parallelization efficiency is obtained for *wireless + column-major* and this is attributed to the high allocation-time

overheads for larger system sizes, proving that this allocation method is less scalable with system size.

5.7.3 Proportion of Flits Using Wireless Shortcuts

Fig. 5.6 shows the percentage of total flits that used the wireless shortcuts. Note that the number of shortcuts (three) is much lower than the number of nodes (64, 256) in the system. As expected, *2D_parallel + wireless*, being a wireless-agnostic allocation method, leads to the lowest proportion of flits using wireless shortcuts. On the other hand, *wireless + column-major* allocation leads

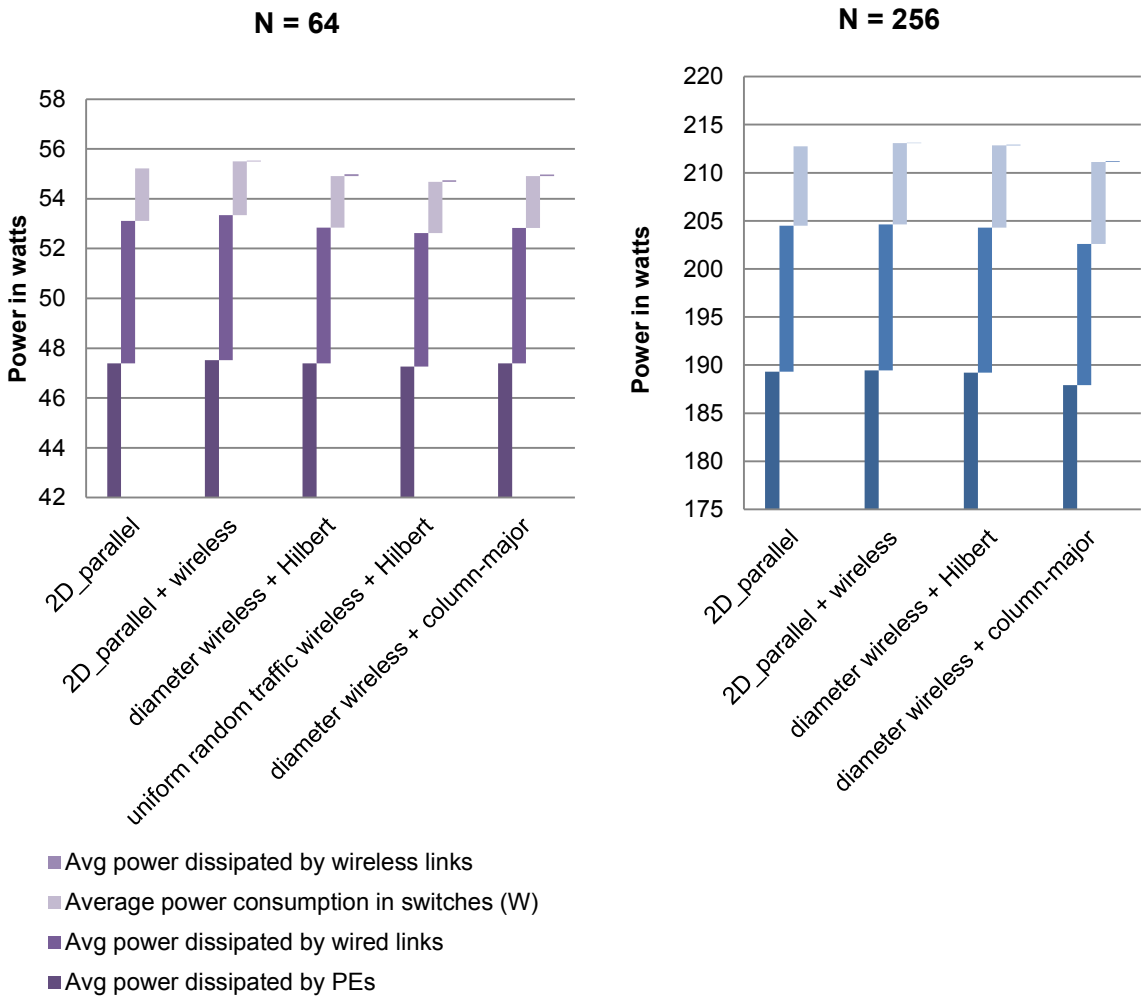


Figure 5-7. Average power dissipation in different NoC architectures and system sizes.

to the highest proportion of flits using wireless shortcuts across system sizes. This is because it is a wireless-aware allocation method, in which the partitions that do not get direct access to wireless shortcuts are aligned with the shortcuts, providing them with access to the shortcuts during routing, as explained earlier in Section 5.5.3.

5.7.4 Energy and Power Consumption

Average power dissipation in the chip is important from the physical perspective, because it is a direct indicator of the activity of the logic inside the chip and has a bearing on its thermal profile as explained further in Section 5.7.6. Quite predictably, the average power dissipation is higher in architectures that can deliver higher computation throughput, as shown in Fig. 5.7. In fact, wireless-aware allocation consistently leads to lower average power dissipation. We have included a data point to show that wireless link placement under uniform random traffic assumption leads to even lower power dissipation for N=64, although this is primarily because fewer computations are being performed and fewer messages are being transferred per second. Note that, the reduced power dissipation comes at the cost of reduced throughput performance in all cases. Consequently, we evaluated the energy consumption profiles of the architectures under consideration.

In order to determine which architecture is indeed the most energy-efficient, we evaluated the energy spent per operation. This consists of the computation energy component spent within the computation nodes, and the network energy component spent in the network switches, wireless transceivers and wired links. Fig. 5.6 shows a comparison of the energy spent per operation across different network architectures and system sizes. *2D_parallel + wireless* is the most energy-efficient in terms of overall energy consumption per operation. A closer look reveals that for N=64, the network energy component is

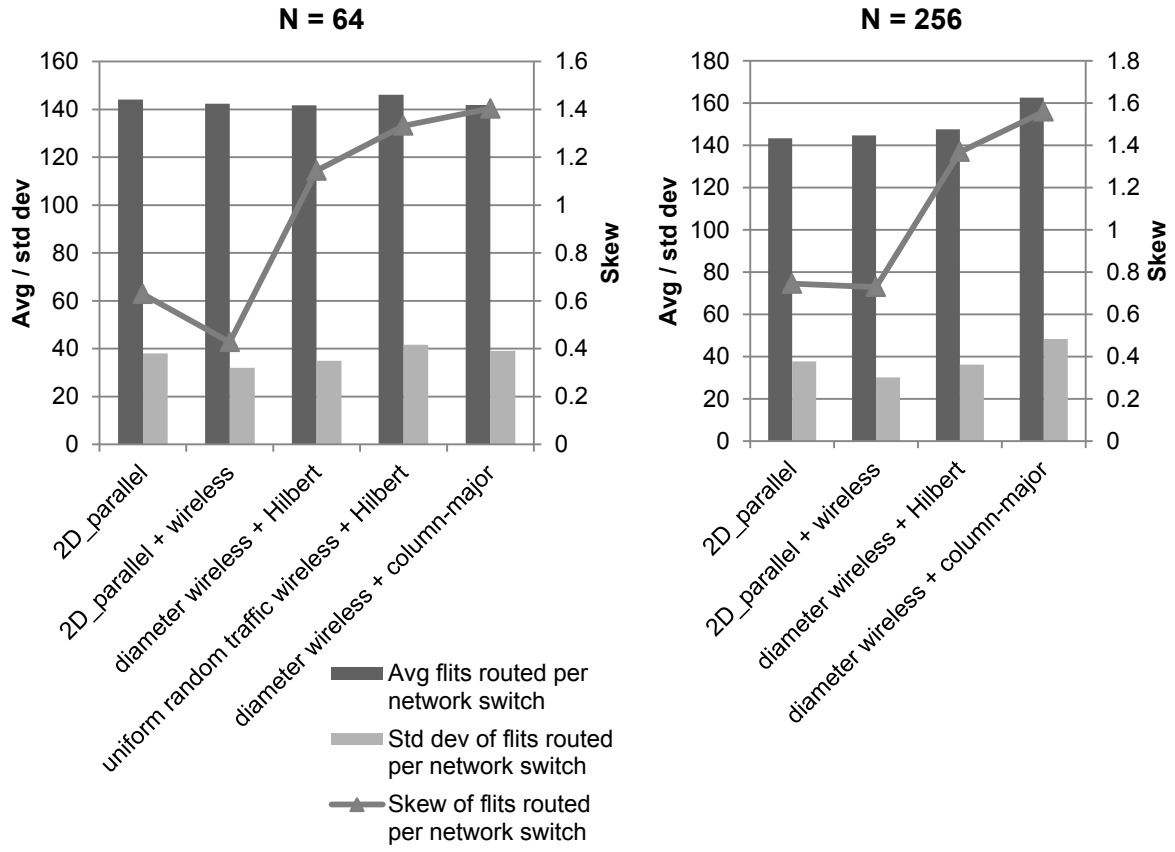


Figure 5-8. Average, standard deviation and skew of flits routed per network switch across NoC architectures and system sizes.

indeed lower for the wireless-aware methods, *wireless + Hilbert* and *wireless + column-major*, due to a larger proportion of their flits using wireless shortcuts, each of which consumes less energy than a wired link. However, due to higher

computation latencies and the greater contribution of the computation energy component, the overall energy per operation turns out to be higher. For N=256, the proportion of flits using wireless shortcuts is low across all architectures, and the saving in energy due to flits using wireless shortcuts is more than offset by the additional energy consumption in the wired links. This leads us to the conclusion that *2D_parallel + wireless* is still the best performing architecture from the energy-efficiency perspective.

5.7.5 Traffic Statistics

In order to thoroughly analyze the throughput and energy-efficiency of different architectures, we need to understand the nature of traffic that our application generates. Our use-case model does not generate a deterministic traffic pattern. Hence, we try to characterize the traffic in terms of its first, second and third order statistical properties, and correlate these with throughput, energy consumption and power dissipation. A good indicator of traffic is the number of flits routed per network switch while running the application. We measure the mean, standard deviation and skew of this quantity across all 64 (256) switches for N=64 (N=256), as shown in Fig. 5.8. For N=64, the mean values are about the same across architectures; for N=256, *diameter wireless + column-major* clearly needs to route more flits per network switch, which indicates congestion and hence reduced throughput as we have seen earlier. Note that the standard deviation varies across architectures for both system sizes, and is the least for *2D_parallel + wireless*, which has the highest throughput and lowest energy per operation. Traffic is clearly less skewed for wireless-agnostic architectures than for wireless-aware architectures. This is

attributable to congestion around shortcuts in wireless-aware architectures, as discussed earlier. Higher skew is strongly correlated with lower throughput and higher energy per operation. Following the discussion in Section 5.7.4, higher skew is also correlated with lower power dissipation owing to reduced network and PE activity.

5.7.6 Thermal Profile

Thermal profiling of a many-core chip is important in order to prevent chip failure due to extreme temperatures during periods of peak activity. It is also important to ensure that a large number of hotspots are not created and on-chip temperature variation is low enough not to introduce timing failures. With this objective, we used HotSpot 5.0 [122] to carry out thermal profiling of our systems with N=64 to determine the relationship between NoC architecture and on-chip thermal variation. As shown in Fig. 5.7, the majority of the power dissipation is due to computation activity in the PEs. Hence, the method of allocating these PEs to different jobs has a direct bearing on thermal variation and hotspot creation.

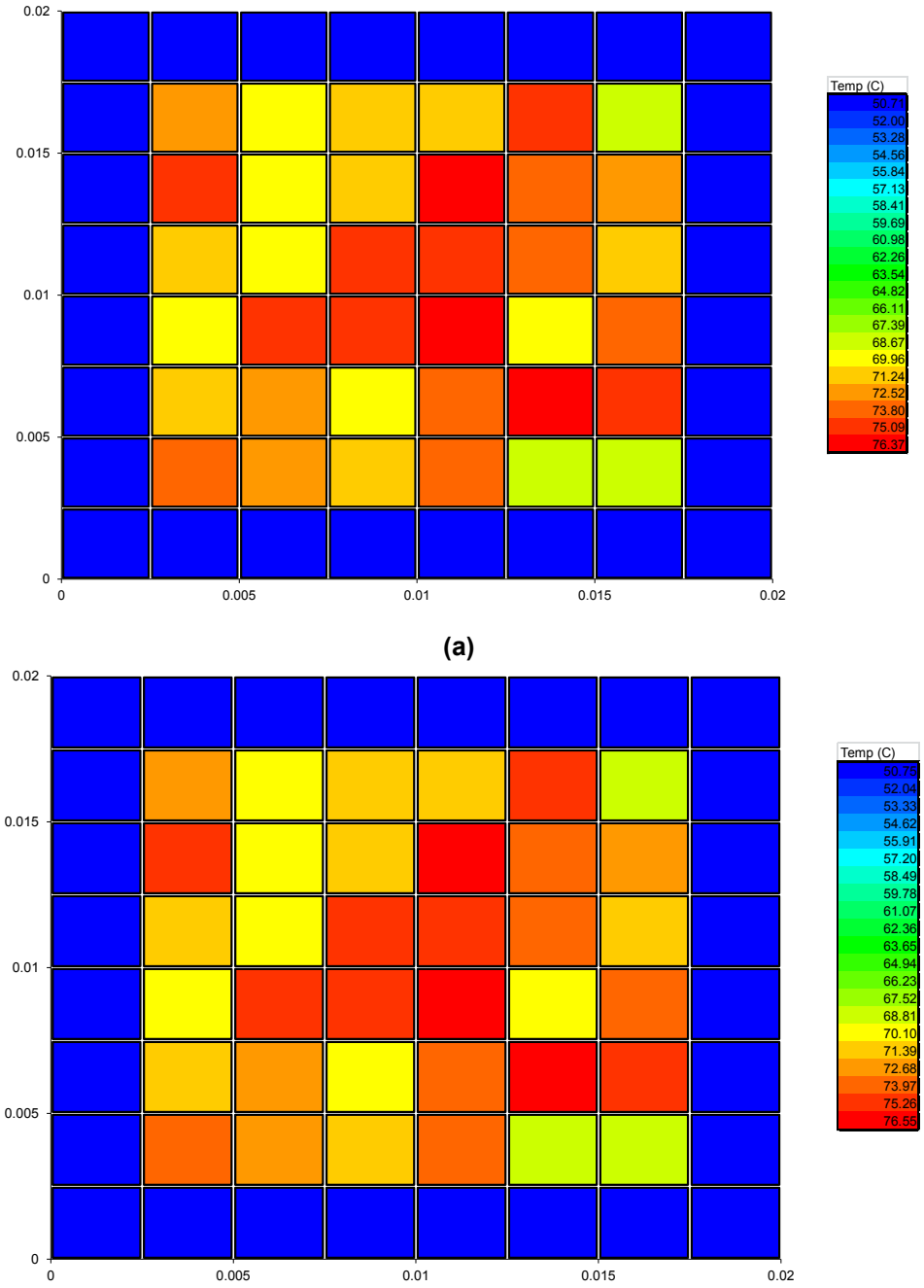


Figure 5-9. Thermal profile of *N*=64 systems with (a) *2D_parallel* and (b) *2D_parallel + wireless* architecture.

Fig. 5.9 (a) and (b) respectively shows thermal profiles for *2D_parallel* and *2D_parallel + wireless* that have a similar pattern albeit for a slightly higher peak temperature in the latter case. Although the maximum temperature (< 77 degrees C) is well within reliability limits, a clustering of hotspots is noticed.

Compare this with the thermal profiles of systems having a wireless-aware architecture (Fig. 5.10). Since average power dissipation is lower in these cases, the thermal profile indicates a more even distribution of hotspots,

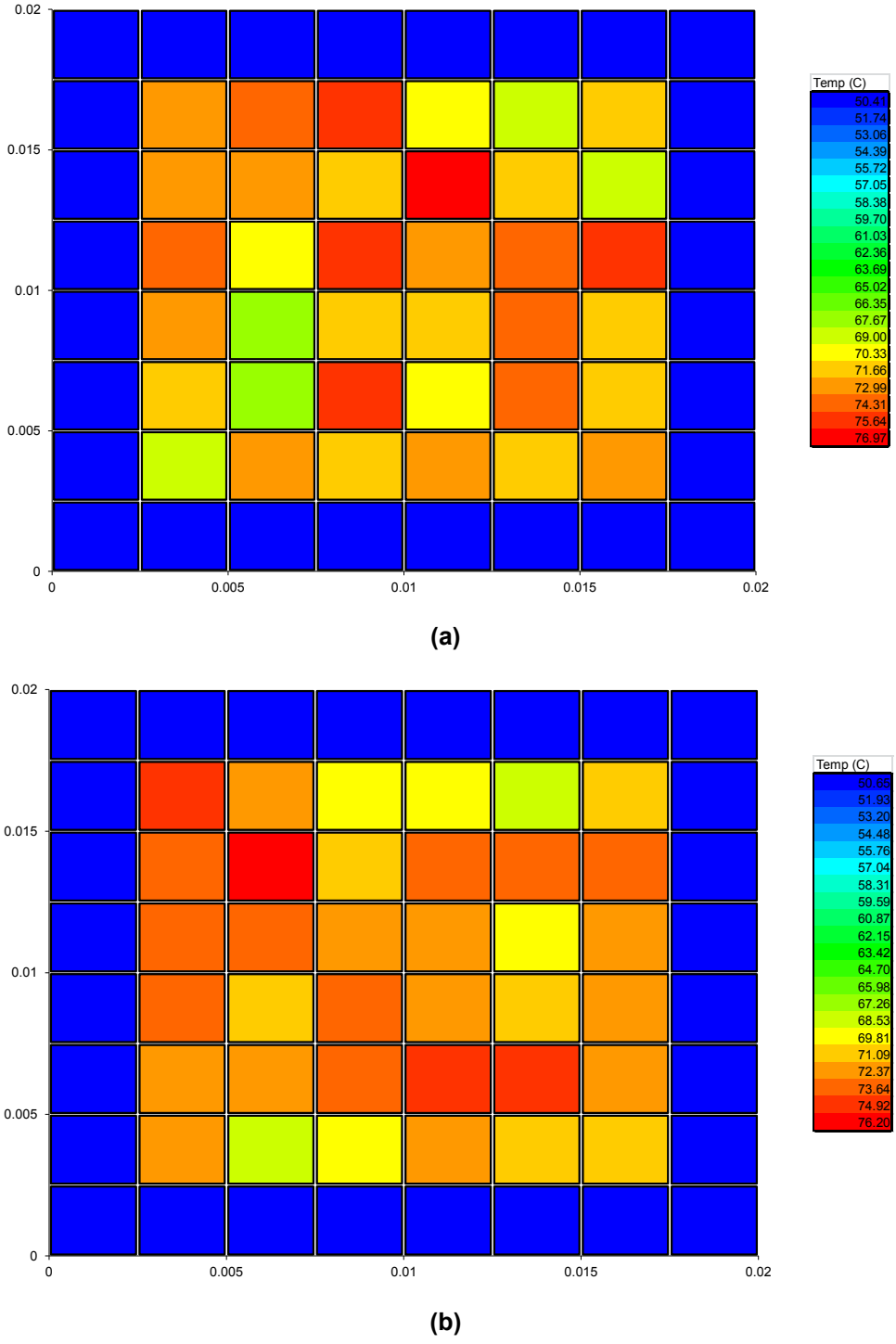


Figure 5-10. Thermal profile of *N*=64 systems with (a) *diameter wireless* + *Hilbert* and (b) *diameter wireless* + *column-major* architecture.

although the on-chip range of temperature is similar to that seen in Fig. 5.9.

This observation can be expected as we found the wireless-aware architectures to compromise on throughput and thus dissipate lower average power, which naturally translates to a lower probability of hotspot generation. The system designer has to decide the tradeoff between higher, energy-efficient throughput on one hand, and lower propensity for hotspot creation on the other, while choosing the NoC architecture and PE allocation approach.

5.8 Conclusion

In this chapter, we propose, design and evaluate novel NoC-based hardware accelerator platforms targeted towards high-throughput scientific applications. The on-chip network is built using both wired links and on-chip wireless links, the latter being used as long-range shortcuts to further reduce inter-core message latency. In addition to achieving high-throughput, we show that our many-core accelerator platforms are energy-efficient.

We achieved computation throughput of over 10^{11} log/exp operations per second for a class of scientific applications involving concurrently-executing jobs of similar nature but variable computational footprint, while consuming ~0.5 nJ for each such operation. Our systems dissipate 55 W (for N=64) and 213 W (for N=256) and the maximum on-chip temperature is capped at 77 degrees C, demonstrating that high throughput is achieved without sacrificing energy-efficiency or exceeding power and thermal budgets, thereby being thermally efficient. We analyze the traffic behavior through statistical properties and correlate these with observations of throughput performance and power dissipation. We explore several NoC architectures and evaluate them with

respect to the above-mentioned parameters and present design tradeoffs between throughput and energy-efficiency, and on-chip thermal variation. The results presented herein provide solutions to several challenges a system architect faces when designing low-energy high-performance many-core hardware accelerators.

6. Conclusion and Future Research

The aim of this doctoral work has been to demonstrate the potential of NoC-based many-core systems to act as enablers for complex computational biology applications. To the best of our knowledge, this represents the first comprehensive work undertaken to leverage the NoC paradigm for a high-performance scientific computing application. The platforms proposed and designed as part of this work have been demonstrated to deliver orders of magnitude superior performance when compared with other hardware platforms. The results are promising, and they open up the scope for further research on NoC-based platforms having novel on-chip interconnects and architectures, distributed cores and memories on a chip, and a broader application space.

6.1 NoC-based Platforms for Biocomputing: A Ready-Reckoner

In this dissertation, we have shown how one can harness the power of on-chip network-enabled many-core architectures to enable time and energy-efficient solutions to complex computational biology problems. This opens up the scope for further research in this area geared towards solving problems with similar computational characteristics but derived from other scientific domains. In the following, we distill our contributions in this dissertation and provide a so-called ready-reckoner in Table 7, which we believe will go a long way in providing guidelines to the designer in making appropriate choices and decisions while architecting NoC-based platforms to target the problem at hand. The organization of the table is as follows. Each of the rows specifies a hardware or software design parameter. Each column refers to an application class along with

the defining characteristics. Each entry in the table mentions the choice(s) that delivered optimal performance in our study.

Table 7. Ready-reckoner for NoC-based platform design targeting computational biology applications

For sequence analysis, see [114]. For details on the rest, refer to Chapters 3, 4 and 5. The input size (number of DNA characters in sequence analysis, number of genes in branch-and-bound) is denoted by *n*. The number of processing elements (PEs) is *p*. HW/SW design parameters are in green. HW design parameters are in dark red. SW design parameters are in dark blue.

	Combinatorial optimization		***Throughput-driven computation*** ↓ ***ML phylogeny tree optimization***
	Sequence analysis (see [114])	***Exhaustive search (e.g. branch-and-bound)***	
Memory footprint	$O(n/p)$ $n \sim 10^2$ - 10^4	$O(n^2)$ $n \sim 10^2$	$O(1)$
Associated traffic pattern	hypercubic, mesh	broadcast	point-to-point (arbitrary)
Network topology	mesh / torus with switch bypass	quad-tree mesh / torus	mesh / torus
On-chip interconnection technology	wired	wired	wired-only wired + long-range wireless
Chip integration dimensionality	2-D	2-D	2-D, 3-D
Processing element architecture	custom lightweight integer	custom lightweight integer	custom lightweight floating-point
Task allocation policy	-	task granularity (optimal subtree rooting)	* Space-filling curve (e.g. Hilbert curve) based for locality preservation * Serial first-fit / parallel best-fit * Wireless agnostic / aware
Data mapping	block decomposition	-	-
Message routing policy	structured, regular communication	conditional broadcast	wormhole e-cube routing (with multiple virtual channels)

6.2 Architecture Space Exploration

Most real-world scientific applications consist of smaller task kernels concurrently running with variable computation footprints. The distribution of these kernels – varying across application classes – is usually neither completely regular nor totally random. The degree of hardware acceleration depends on the degree of connectivity among the cores. Regular topologies prove to be inadequate when dealing with such scenarios because multi-hop core-to-core communication impacts both latency and energy consumption. Novel interconnection architectures based on *Small-World Graphs* have been shown to be very successful in reducing network diameters in graphs with many nodes. The Internet, social networks and network of neurons in the human brain are examples of graphs having Small-World property. Such networks consist of a combination of short-range (next-to-neighbor) and long-range links. Analyses of classes of applications would help in generating the Small-World network that fits the application traffic. There is a tradeoff involved in choosing the best-fit network architecture for a particular application and maintaining its reusability in a broader application class.

Implementation of such Small-World networks on a chip is still a challenge because traditional on-chip metal wires as long-range links do not provide appreciable improvements in latency or energy consumption because long copper wires introduce significant delay and power consumption. Novel interconnects such as RF [119], wireless [120] or photonic [121] links have hence been proposed. Carbon nanotube based on-chip THz wireless links have been considered in this work (Chapter 5). These novel interconnects hold promise in

providing high-bandwidth (low-latency) and low-power on-chip long-range links. Research in this area would be able to leverage the advances in novel interconnect technology coupled with novel network topologies to build NoC-driven many-core platforms with higher throughput capabilities.

6.3 Application Space Exploration

Biocomputing applications targeted in this work include Maximum Parsimony and Maximum Likelihood phylogeny reconstruction. We have also demonstrated the potential of NoC-based platforms to cater to throughput-oriented applications. In the field of phylogenetics, one important application could be Bayesian Inference (BI) [123], [124], which is computationally similar to Maximum Likelihood. Other HPC applications – be it climate modeling or advanced materials science research – could potentially benefit from NoC-based platforms. With the increasing diversity of applications, a large number of such platforms would consist of heterogeneous cores and/or distributed processor and memory nodes. As such, the interconnect fabric would need to support newer kinds of inter-node communication. There is enough indication that *distributed processor-memory interactions within a chip* would present the greatest bottlenecks to throughput in systems of the future, and current research on NoC is beginning to focus on this, e.g. [125]. With applications becoming more computation- and data-intensive, each memory-processor interface in a distributed many-core system needs to match or better cache speeds attainable today. This is the primary motivation behind long-term research on interconnect topologies with a focus to solve large problems of the future on a chip. The rationale behind this goal is that with power consumption being an overriding

concern in the same manner as speed was a decade ago, more solutions would increasingly be sought to be implementable on a chip, as opposed to clusters or supercomputers. It also helps to note that on-chip solutions would often be faster and cost less, and research in this area would be a great enabler. The sheer variety of scientific applications and their application traffic – both among processing nodes, and between processing nodes and memories – presents an interesting field of study to a researcher on multi-core systems and on-chip interconnects. Clearly, activity in this research space would yield *low-power high-performance computing systems on a chip.*

7. References

[1] D. A. Bader and M. Yan, “High-Performance Phylogeny Reconstruction” in Handbook of Computational Molecular Biology, Edited by S. Aluru, Chapman & Hall/CRC Computer and Information Science Series, 2005.

[2] P.H. Harvey and M.D. Pagel. The Comparative Method in Evolutionary Biology. Oxford University Press, 1991.

[3] M. Blanchette, G. Bourque, and D. Sankoff, “Breakpoint phylogenies,” Genome Informatics Workshop, Tokyo: University Academy Press, 1997, pp. 25-34.

[4] E.L. Lawler, J. Lenstra, A.R. Kan and D. Shmoys. The traveling salesman problem. John Wiley, 1985.

[5] I. Pe'er and R. Shamir, “The median problems for breakpoints are NP-complete,” Elec. Colloq. on Comput. Complexity, 1998, p. 71.

[6] T. Jukes, C. Cantor, “Evolution of protein molecules”, Mammalian protein metabolism, III:21–132, Academic Press, New York, 1969.

[7] M. Kimura. A simple method for estimating evolutionary rates of base substitutions by thorough comparative studies of nucleotide sequences”, J. Mol. Evol., 16:111-120, 1980.

[8] M. Hasegawa, H. Kishino, T. Yano, “Dating of the human-ape splitting by a molecular clock of mitochondrial DNA”, J. Mol. Evol., 22:160–174, 1985.

[9] C. Lanave et al, “A new method for calculating evolutionary substitution rates”, J. Mol. Evol., 20:86–93, 1984.

[10] F. Rodriguez, et al, “The general stochastic model of nucleotide substitution”, J. Theor. Biol., 142:485–501, 1990.

[11] Felsenstein, J. 1981. Evolutionary trees from DNA sequences: A maximum likelihood approach. J. Molecular Evolution 17, pp. 368-376.

[12] Felsenstein, J. 2004. *Inferring Phylogenies*. Sinauer Associates, Inc.

[13] Chor, B. and Tuller, T. 2005. Maximum Likelihood of Evolutionary Trees: Hardness and Approximation. *Bioinformatics*, vol. 21(1), pp. 97-106.

[14] Bakos, J.D.; Elenis, P.E.; , "A Special-Purpose Architecture for Solving the Breakpoint Median Problem," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* , vol.16, no.12, pp.1666-1676, Dec. 2008.

[15] J. Bentley. Fast algorithms for geometric traveling salesman problems. ORSA Journal of Computing, 4:387-411, 1992.

[16] B. Golden, L. Bodin, T. Doyle, W. Stewart. Approximate traveling salesman algorithms, Operations Research, 28:694-711, 1980.

[17] G. Reinelt. The traveling salesman problem: computational solutions for TSP applications. In LNCS 840, pp. 172-186, Springer-Verlag, Berlin, 1994.

[18] S. Lin and B. Kernighan. An effective heuristic algorithm for the traveling salesman problem. Operations Research, 21:498-516, 1973.

[19] P. Jog, J. Y. Suh, and D. Van Gucht. Parallel Genetic Algorithms Applied to the Traveling Salesman Problem, SIAM Journal of Optimization, 1(4): 515-529, 1991.

[20] D. L. Miller, J. F. Pekny. Results from a parallel branch and bound algorithm for the asymmetric traveling salesman problem, Operations Research Letters, 8(3): 129-135, 1989.

[21] M. Bellmore and G. Nemhauser, "The Traveling Salesman Problem: A Survey," Operations Research, 16: 538-558, 1968.

[22] E. Horowitz and S. Sahni, "Branch-and-bound" in Fundamentals of computer algorithms, Potomac, MD: Computer Science Press, 1984, pp. 370-421.

[23] Dally, W.J.; Towles, B.; , "Route packets, not wires: on-chip interconnection networks," *Design Automation Conference, 2001. Proceedings* , vol., no., pp. 684-689, 2001.

[24] Vangal, S.R.; Howard, J.; Ruhl, G.; Dighe, S.; Wilson, H.; Tschanz, J.; Finan, D.; Singh, A.; Jacob, T.; Jain, S.; Erraguntla, V.; Roberts, C.; Hoskote, Y.; Borkar, N.; Borkar, S.; , "An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS," *Solid-State Circuits, IEEE Journal of* , vol.43, no.1, pp.29-41, Jan. 2008.

[25] Henkel, J.; Wolf, W.; Chakradhar, S.; , "On-chip networks: a scalable, communication-centric embedded system design paradigm," *VLSI Design, 2004. Proceedings. 17th International Conference on* , vol., no., pp. 845- 851, 2004.

[26] Tobias Bjerregaard and Shankar Mahadevan. 2006. A survey of research and practices of Network-on-chip. *ACM Comput. Surv.* 38, 1, Article 1 (June 2006).

[27] Marculescu, R.; Ogras, U.Y.; Li-Shiuan Peh; Jerger, N.E.; Hoskote, Y.; , "Outstanding Research Problems in NoC Design: System, Microarchitecture, and Circuit Perspectives," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* , vol.28, no.1, pp.3-21, Jan. 2009.

[28] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Mateo, CA: Morgan Kaufmann, 2004.

[29] Varatkar, G.V.; Marculescu, R.; , "On-chip traffic modeling and synthesis for MPEG-2 video applications," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* , vol.12, no.1, pp.108-119, Jan. 2004.

[30] Soteriou, V.; Hangsheng Wang; Peh, L.; , "A Statistical Traffic Model for On-Chip Interconnection Networks," *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2006. MASCOTS 2006. 14th IEEE International Symposium on* , vol., no., pp. 104- 116, 11-14 Sept. 2006.

[31] Grecu, C.; Ivanov, A.; Pande, R.; Jantsch, A.; Salminen, E.; Ogras, U.; Marculescu, R.; , "Towards Open Network-on-Chip Benchmarks," *Networks-on-Chip, 2007. NOCS 2007. First International Symposium on* , vol., no., pp.205, 7-9 May 2007.

[32] Cohen, I.; Rottenstreich, O.; Keslassy, I.; , "Statistical Approach to Networks-on-Chip," *Computers, IEEE Transactions on* , vol.59, no.6, pp.748-761, June 2010.

[33] Jingcao Hu; Marculescu, R.; , "Energy-aware mapping for tile-based NoC architectures under performance constraints," *Design Automation Conference, 2003. Proceedings of the ASP-DAC 2003. Asia and South Pacific* , vol., no., pp. 233- 239, 21-24 Jan. 2003.

[34] Srinivasan, K.; Chatha, K.S.; , "A technique for low energy mapping and routing in network-on-chip architectures," *Low Power Electronics and Design, 2005. ISLPED '05. Proceedings of the 2005 International Symposium on* , vol., no., pp. 387- 392, 8-10 Aug. 2005.

[35] Murali, S.; Meloni, P.; Angiolini, F.; Atienza, D.; Carta, S.; Benini, L.; De Micheli, G.; Raffo, L.; , "Designing Application-Specific Networks on Chips with Floorplan Information," *Computer-Aided Design, 2006. ICCAD '06. IEEE/ACM International Conference on* , vol., no., pp.355-362, 5-9 Nov. 2006.

[36] Srinivasan Murali; Coenen, M.; Radulescu, A.; Goossens, K.; De Micheli, G.; , "A Methodology for Mapping Multiple Use-Cases onto Networks on Chips," *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings* , vol.1, no., pp.1-6, 6-10 March 2006.

[37] Chen-Ling Chou; Marculescu, R.; , "Contention-aware application mapping for Network-on-Chip communication architectures," *Computer Design, 2008.*

ICCD 2008. IEEE International Conference on , vol., no., pp.164-169, 12-15 Oct. 2008.

[38] Bakhoda, A.; Kim, J.; Aamodt, T.M.; , "Throughput-Effective On-Chip Networks for Manycore Accelerators," *Microarchitecture (MICRO), 2010 43rd Annual IEEE/ACM International Symposium on* , vol., no., pp.421-432, 4-8 Dec. 2010.

[39] Trivino, F.; Sanchez, J.L.; Alfaro, F.J.; Flich, J.; , "Exploring NoC Virtualization Alternatives in CMPs," *Parallel, Distributed and Network-Based Processing (PDP), 2012 20th Euromicro International Conference on* , vol., no., pp.473-482, 15-17 Feb. 2012.

[40] Pop, P.; Eles, P.; Pop, T.; Peng, Z.; , "An approach to incremental design of distributed embedded systems," *Design Automation Conference, 2001. Proceedings* , vol., no., pp. 450- 455, 2001.

[41] Yuan Xie; Wolf, W.; , "Allocation and scheduling of conditional task graph in hardware/software co-synthesis," *Design, Automation and Test in Europe, 2001. Conference and Exhibition 2001. Proceedings* , vol., no., pp.620-625, 2001.

[42] Jiong Luo; Jha, N.K.; , "Power-conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems," *Computer Aided Design, 2000. ICCAD-2000. IEEE/ACM International Conference on* , vol., no., pp.357-364, 2000.

[43] Varatkar, G.; Marculescu, R.; , "Communication-aware task scheduling and voltage selection for total systems energy minimization," *Computer Aided*

Design, 2003. ICCAD-2003. International Conference on , vol., no., pp. 510- 517, 9-13 Nov. 2003.

[44] Chong Sun, Li Shang, and Robert P. Dick. 2007. Three-dimensional multiprocessor system-on-chip thermal optimization. In *Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis* (CODES+ISSS '07). ACM, New York, NY, USA, 117-122.

[45] Pham, D.; Asano, S.; Bolliger, M.; Day, M.N.; Hofstee, H.P.; Johns, C.; Kahle, J.; Kameyama, A.; Keaty, J.; Masubuchi, Y.; Riley, M.; Shippy, D.; Stasiak, D.; Suzuoki, M.; Wang, M.; Warnock, J.; Weitzel, S.; Wendel, D.; Yamazaki, T.; Yazawa, K.; , "The design and implementation of a first-generation CELL processor," *Solid-State Circuits Conference, 2005. Digest of Technical Papers. ISSCC. 2005 IEEE International* , vol., no., pp.184-592 Vol. 1, 10-10 Feb. 2005.

[46] Gratz, P.; Changkyu Kim; McDonald, R.; Keckler, S.W.; Burger, D.; , "Implementation and Evaluation of On-Chip Network Architectures," *Computer Design, 2006. ICCD 2006. International Conference on* , vol., no., pp.477-484, 1-4 Oct. 2006.

[47] Srinivasan, K.; Chatha, K.S.; , "A Low Complexity Heuristic for Design of Custom Network-on-Chip Architectures," *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings* , vol.1, no., pp.1-6, 6-10 March 2006.

[48] Song, Z.; Ma, G.; Song, D.; , "Hierarchical Star: An Optimal NoC Topology for High-Performance SoC Design," *Computer and Computational Sciences,*

2008. IMSCCS '08. International Multisymposiums on , vol., no., pp.158-163, 18-20 Oct. 2008.

[49] Balkan, A.O.; Gang Qu; Vishkin, U.; , "Mesh-of-Trees and Alternative Interconnection Networks for Single-Chip Parallelism," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* , vol.17, no.10, pp.1419-1432, Oct. 2009.

[50] Camacho, J.; Flich, J.; , "HPC-Mesh: A Homogeneous Parallel Concentrated Mesh for Fault-Tolerance and Energy Savings," *Architectures for Networking and Communications Systems (ANCS), 2011 Seventh ACM/IEEE Symposium on* , vol., no., pp.69-80, 3-4 Oct. 2011.

[51] Yu-Hsiang Kao; Ming Yang; Artan, N.S.; Chao, H.J.; , "CNoC: High-Radix Clos Network-on-Chip," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* , vol.30, no.12, pp.1897-1910, Dec. 2011.

[52] Dally, W.J.; Seitz, C.L.; , "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *Computers, IEEE Transactions on* , vol.C-36, no.5, pp.547-553, May 1987.

[53] Nilsson, E.; Millberg, M.; Oberg, J.; Jantsch, A.; , "Load distribution with the proximity congestion awareness in a network on chip," *Design, Automation and Test in Europe Conference and Exhibition, 2003* , vol., no., pp. 1126- 1127, 2003.

[54] DaeHo Seo; Akif Ali; Won-Taek Lim; Rafique, N.; , "Near-optimal worst-case throughput routing for two-dimensional mesh networks," *Computer*

Architecture, 2005. ISCA '05. Proceedings. 32nd International Symposium on , vol., no., pp. 432- 443, 4-8 June 2005.

[55] Pamunuwa, D.;Öberg, J; Zheng, L. R.; Millberg, M.; Jantsch, A.; Tenhunen, H.; , "Layout, Performance and Power Trade-Offs in Mesh-Based Network-on-Chip Architectures," *Very Large Scale Integration (VLSI-SoC) 2003. Proceedings of the 12th IFIP International Conference on,* pp. 362-367.

[56] Seiculescu, C.; Murali, S.; Benini, L.; De Micheli, G.; , "SunFloor 3D: A Tool for Networks on Chip Topology Synthesis for 3-D Systems on Chips," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* , vol.29, no.12, pp.1987-2000, Dec. 2010.

[57] Bainbridge, W.J.; Furber, S.B.; , "Delay insensitive system-on-chip interconnect using 1-of-4 data encoding," *Asynchronus Circuits and Systems, 2001. ASYNC 2001. Seventh International Symposium on* , vol., no., pp.118-126, 2001.

[58] Ludovici, D.; Strano, A.; Gaydadjiev, G.N.; Benini, L.; Bertozzi, D.; , "Design space exploration of a mesochronous link for cost-effective and flexible GALS NOCs," *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010* , vol., no., pp.679-684, 8-12 March 2010.

[59] Mandal, A.; Khatri, S.P.; Mahapatra, R.N.; , "A fast, source-synchronous ring-based network-on-chip design," *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012* , vol., no., pp.1489-1494, 12-16 March 2012.

[60] Salamy, H.; Harmanani, H.; , "An effective solution to thermal-aware test scheduling on network-on-chip using multiple clock rates," *Circuits and Systems (MWSCAS), 2012 IEEE 55th International Midwest Symposium on* , vol., no., pp.530-533, 5-8 Aug. 2012.

[61] Li Shang; Li-Shiuan Peh; Jha, N.K.; , "Dynamic voltage scaling with links for power optimization of interconnection networks," *High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings. The Ninth International Symposium on* , vol., no., pp. 91- 102, 8-12 Feb. 2003.

[62] Soteriou, V.; Li-Shiuan Peh; , "Exploring the Design Space of Self-Regulating Power-Aware On/Off Interconnection Networks," *Parallel and Distributed Systems, IEEE Transactions on* , vol.18, no.3, pp.393-408, March 2007.

[63] Ogras, U.Y.; Marculescu, R.; Marculescu, D.; Eun Gu Jung; , "Design and Management of Voltage-Frequency Island Partitioned Networks-on-Chip," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* , vol.17, no.3, pp.330-341, March 2009.

[64] Liang Guang; Liljeberg, P.; Nigussie, E.; Tenhunen, H.; , "A review of dynamic power management methods in NoC under emerging design considerations," *NORCHIP, 2009* , vol., no., pp.1-6, 16-17 Nov. 2009.

[65] Bertozzi, D.; Benini, L.; De Micheli, G.; , "Error control schemes for on-chip communication links: the energy-reliability tradeoff," *Computer-Aided Design of*

Integrated Circuits and Systems, IEEE Transactions on , vol.24, no.6, pp. 818-831, June 2005.

[66] Hui Zhao; Kandemir, M.; Irwin, M.J.; , "Exploring performance-power tradeoffs in providing reliability for NoC-based MPSoCs," *Quality Electronic Design (ISQED), 2011 12th International Symposium on* , vol., no., pp.1-7, 14-16 March 2011.

[67] Vitkovskiy, A.; Soteriou, V.; Nicopoulos, C.; , "A Dynamically Adjusting Gracefully Degrading Link-Level Fault-Tolerant Mechanism for NoCs," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* , vol.31, no.8, pp.1235-1248, Aug. 2012.

[68] Mak, T.S.T.; Lam, K.P.; , "High speed GAML-based phylogenetic tree reconstruction using HW/SW codesign," *Bioinformatics Conference, 2003. CSB 2003. Proceedings of the 2003 IEEE* , vol., no., pp. 470- 473, 11-14 Aug. 2003.

[69] Alachiotis, N.; Sotiriades, E.; Dollas, A.; Stamatakis, A.; , "Exploring FPGAs for accelerating the phylogenetic likelihood function," *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on* , vol., no., pp.1-8, 23-29 May 2009.

[70] Stamatakis, A. 2006. RAxML-VI-HPC: Maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics*.

[71] Blagojevic, F.; Stamatakis, A.; Antonopoulos, C.D.; Nikolopoulos, D.S.; , "RAxML-Cell: Parallel Phylogenetic Tree Inference on the Cell Broadband

Engine," *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International* , vol., no., pp.1-10, 26-30 March 2007.

[72] S. Zierke and J.D. Bakos, “FPGA Acceleration of the phylogenetic likelihood function for Bayesian MCMC inference methods,” *BMC Bioinformatics*, 11: 184: 1-12, 2010.

[73] Pratas, F.; Trancoso, P.; Stamatakis, A.; Sousa, L.; , "Fine-grain Parallelism Using Multi-core, Cell/BE, and GPU Systems: Accelerating the Phylogenetic Likelihood Function," *Parallel Processing, 2009. ICPP '09. International Conference on* , vol., no., pp.9-17, 22-25 Sept. 2009.

[74] Topol, A. W.; Tulipe, D. C. La; Shi, L.; Frank, D. J.; Bernstein, K.; Steen, S. E.; Kumar, A.; Singco, G. U.; Young, A. M.; Guarini, K. W.; Ieong, M.; , "Three-dimensional integrated circuits," *IBM Journal of Research and Development* , vol.50, no.4.5, pp.491-506, July 2006.

[75] Jacob, P.; Erdogan, O.; Zia, A.; Belemjian, P.M.; Kraft, R.P.; McDonald, J.F.; , "Predicting the performance of a 3D processor-memory chip stack," *Design & Test of Computers, IEEE* , vol.22, no.6, pp. 540- 547, Nov.-Dec. 2005.

[76] Pavlidis, V.F.; Friedman, E.G.; , "3-D Topologies for Networks-on-Chip," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* , vol.15, no.10, pp.1081-1090, Oct. 2007.

[77] Feero, B.S.; Pande, P.P.; , "Networks-on-Chip in a Three-Dimensional Environment: A Performance Evaluation," *Computers, IEEE Transactions on* , vol.58, no.1, pp.32-45, Jan. 2009.

[78] Shan Yan; Bill Lin; , "Design of application-specific 3D Networks-on-Chip architectures," *Computer Design, 2008. ICCD 2008. IEEE International Conference on* , vol., no., pp.142-149, 12-15 Oct. 2008.

[79] Yuh-Fang Tsai; Feng Wang; Yuan Xie; Vijaykrishnan, N.; Irwin, M.J.; , "Design Space Exploration for 3-D Cache," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* , vol.16, no.4, pp.444-455, April 2008.

[80] Young-Su Kwon; In-Cheol Park; Chong-Min Kyung; , "A hardware accelerator for the specular intensity of Phong illumination model in 3-dimensional graphics," *Design Automation Conference, 2000. Proceedings of the ASP-DAC 2000. Asia and South Pacific* , vol., no., pp.559-564, 9-9 June 2000.

[81] Abed, K.H.; Siferd, R.E.; , "CMOS VLSI implementation of a low-power logarithmic converter," *Computers, IEEE Transactions on* , vol.52, no.11, pp. 1421- 1433, Nov. 2003.

[82] Li, R.-C.; , "Near optimality of Chebyshev interpolation for elementary function computations," *Computers, IEEE Transactions on* , vol.53, no.6, pp. 678-687, June 2004.

[83] Byeong-Gyu Nam; Hyejung Kim; Hoi-Jun Yoo; , "Power and Area-Efficient Unified Computation of Vector and Elementary Functions for Handheld 3D Graphics Systems," *Computers, IEEE Transactions on* , vol.57, no.4, pp.490-504, April 2008.

[84] Strollo, A.G.M.; De Caro, D.; Petra, N.; , "Elementary Functions Hardware Implementation Using Constrained Piecewise-Polynomial Approximations," *Computers, IEEE Transactions on* , vol.60, no.3, pp.418-432, March 2011.

[85] Jijun Tang; Moret, B.M.E.; LiYing Cui; dePamphilis, C.W.; , "Phylogenetic reconstruction from arbitrary gene-order data," *Bioinformatics and Bioengineering, 2004. BIBE 2004. Proceedings. Fourth IEEE Symposium on* , vol., no., pp. 592- 599, 19-21 May 2004.

[86] Kangmin Lee; Se-Joong Lee; Hoi-Jun Yoo; , "Low-power network-on-chip for high-performance SoC design," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* , vol.14, no.2, pp.148-160, Feb. 2006.

[87] Bononi, L.; Concer, N.; , "Simulation and analysis of network on chip architectures: ring, spidergon and 2D mesh," *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings* , vol.2, no., pp.6 pp., 6-10 March 2006.

[88] Circuits Multi-Projects, 46, Avenue Félix Viallet, 38031 GRENOBLE FRANCE (http://cmp.imag.fr/) Last accessed 27 February 2013.

[89] National Center for Biotechnology Information Genbank (http://www.ncbi.nlm.nih.gov/genbank/). Last date accessed: 27 February 2013.

[90] Genome Evolution Laboratory – Mauve Genome Alignment Software (http://asap.ahabs.wisc.edu/mauve/). Last date accessed: 27 February 2013.

[91] R. A. Fisher, “On the Mathematical Foundations of Theoretical Statistics,” *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, Vol. 222, (1922), pp. 309-368.

[92] A. W. F. Edwards and L. L. Cavalli-Sforza. 1964. Reconstruction of evolutionary tress. pp. 67-76 in *Phenetic and Phylogenetic Classification*, ed. V. H. Heywood and J. McNeill. Systematics Association Publ. No. 6, London.

[93] J. Neyman. 1971. Molecular studies of evolution: A source of novel statistical problems. pp. 1-27 in *Statistical Decision Theory and Related Topics*, ed. S. S. Gupta and J. Yackel. Academic Press, New York.

[94] R. L. Kashyap and S. Subas. 1974. Statistical estimation of parameters in a phylogenetic tree using a dynamic model of the substitution process. *Journal of Theoretical Biology* 47:75-101.

[95] Z. Yang. 1995. A space-time process model for the evolution of DNA sequences. *Genetics* 139:993-1005.

[96] D. L. Swofford. 2002. PAUP*. Phylogenetic Analysis Using Parsimony (*and Other Methods). Version 4. Sinauer Associates, Sunderland, Massachusetts.

[97] Felsenstein, J. 1989. PHYLIP -- Phylogeny Inference Package (Version 3.2). *Cladistics* 5: 164-166.

[98] S. Guindon and O. Gascuel. 2003. A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. *Systematic Biology,* 52(5):696-704.

[99] G. J. Olsen, H. Matsuda, R. Hagstrom and R. Overbeek. 1994. fastDNAmL: a tool for construction of phylogenetic trees of DNA sequences using maximum likelihood. *Computer Applications in the Biosciences*, 10(1):41-48.

[100] Michael Ott, Jaroslaw Zola, Alexandros Stamatakis, and Srinivas Aluru. 2007. Large-scale maximum likelihood-based phylogenetic analysis on the IBM BlueGene/L. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing* (SC '07). ACM, New York, NY, USA, , Article 4 , 11 pages..

[101] The Exelixis Lab, Heidelberg Institute for Theoretical Studies, Heidelberg, Germany (http://sco.h-its.org/exelixis/software.html) Last accessed 27 February 2013.

[102] Stamatakis, A.P.; Ludwig, T.; Meier, H.; Wolf, M.J.; , "Accelerating Parallel Maximum Likelihood-Based Phylogenetic Tree Calculations Using Subtree Equality Vectors," *Supercomputing, ACM/IEEE 2002 Conference* , vol., no., pp. 40, 16-22 Nov. 2002.

[103] Bogdan, P.; Marculescu, R.; , "Non-Stationary Traffic Analysis and Its Implications on Multicore Platform Design," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* , vol.30, no.4, pp.508-519, April 2011.

[104] Partha Pratim Pande; Grecu, C.; Jones, M.; Ivanov, A.; Saleh, R.; , "Performance evaluation and design trade-offs for network-on-chip interconnect architectures," *Computers, IEEE Transactions on* , vol.54, no.8, pp. 1025- 1040, Aug. 2005.

[105] J. Duato, S. Yalamanchili, and L. Ni. 2003. *Interconnection Networks. An Engineering Approach*. Ch. 9. Morgan Kaufmann Publishers.

[106] D. Hilbert, “Über die stetige Abbildung einer Linie auf ein Flächenstück”, *Mathematische Annalen*, vol. 38,no. 3, pp. 459-460, 1891.

[107] Seal, S. and Aluru, S. 2007. Chapter 44: Spatial domain decomposition methods for parallel scientific computing. In *Handbook of Parallel Computing: Models, Algorithms and Applications*, (Ed. S. Rajasekaran and J. Reif). Chapman & Hall/CRC Computer and Information Science Series.

[108] Olaf R. P. Bininda-Emonds, Marcel Cardillo, Kate E. Jones, Ross D. E. MacPhee, Robin M. D. Beck, Richard Grenyer, Samantha A. Price, Rutger A. Vos, John L. Gittleman & Andy Purvis. 2007. The delayed rise of present-day mammals. In *Nature* 446: 507-512.

[109] Amazon Elastic Compute Cloud (http://aws.amazon.com/ec2/) Last accessed 27 February 2013.

[110] The CIPRES Science Gateway (http://www.phylo.org/sub_sections/portal/) Last accessed 27 February 2013.

[111] Earth System Modeling Framework (http://www.earthsystemmodeling.org/) Last accessed 27 February 2013.

[112] A. Kalyanaraman, “Algorithms for genome assembly” in *Encyclopedia of Parallel Computing*, ed. D. Padua , Springer Science+Business Media LLC, New York, USA. DOI 10.1007/978-0-387-09766-4, In Press, 2011.

[113] Majumder, T.; Pande, P.; Kalyanaraman, A.; , "Accelerating Maximum Likelihood Based Phylogenetic Kernels Using Network-on-Chip," *Computer*

Architecture and High Performance Computing (SBAC-PAD), 2011 23rd International Symposium on , vol., no., pp.17-24, 26-29 Oct. 2011.

[114] Sarkar, S.; Kulkarni, G.R.; Pande, P.P.; Kalyanaraman, A.; , "Network-on-Chip Hardware Accelerators for Biological Sequence Alignment," *Computers, IEEE Transactions on* , vol.59, no.1, pp.29-41, Jan. 2010.

[115] Ogras, U.Y.; Marculescu, R.; , ""It's a small world after all": NoC performance optimization via long-range link insertion," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* , vol.14, no.7, pp.693-706, July 2006.

[116] D. J. Watts and S. H. Strogatz. 1998. Collective dynamics of 'small-world' networks. *Nature* 393:440–442.

[117] Ganguly, A.; Chang, K.; Deb, S.; Pande, P.P.; Belzer, B.; Teuscher, C.; , "Scalable Hybrid Wireless Network-on-Chip Architectures for Multicore Systems," *Computers, IEEE Transactions on* , vol.60, no.10, pp.1485-1502, Oct. 2011.

[118] K. Kempa, J. Rybczynski, Z. Huang, K. Gregorczyk, A. Vidan, B. Kimball, J. Carlson, G. Benham, Y. Wang, A. Herczynski, Z. F. Ren, "Carbon Nanotubes as Optical Antennae," *Advanced Materials*, vol. 19, issue 3, pp. 421-426, February 2007.

[119] M.-C. Frank Chang, Eran Socher, Sai-Wang Tam, Jason Cong, and Glenn Reinman. 2008. RF interconnects for communications on-chip. In *Proceedings of*

the 2008 international symposium on Physical design (ISPD '08). ACM, New York, NY, USA, 78-83..

[120] Floyd, B.A.; Chih-Ming Hung; O, K.K.; , "Intra-chip wireless interconnect for clock distribution implemented with integrated antennas, receivers, and transmitters," *Solid-State Circuits, IEEE Journal of* , vol.37, no.5, pp.543-552, May 2002.

[121] O'Connor, I.; Tissafi-Drissi, F.; Gaffiot, F.; Dambre, J.; De Wilde, M.; Van Campenhout, J.; Van Thourhout, D.; Stroobandt, D.; , "Systematic Simulation-Based Predictive Synthesis of Integrated Optical Interconnect," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* , vol.15, no.8, pp.927-940, Aug. 2007.

[122] Wei Huang; Sankaranarayanan, K.; Skadron, K.; Ribando, R.J.; Stan, M.R.; , "Accurate, Pre-RTL Temperature-Aware Design Using a Parameterized, Geometric Thermal Model," *Computers, IEEE Transactions on* , vol.57, no.9, pp.1277-1288, Sept. 2008.

[123] B. Rannala, and Z. Yang. 1996. Probability distribution of molecular evolutionary trees: a new method of phylogenetic inference. *J. Mol. Evol.* 43:304-311.

[124] Z. Yang and B. Rannala. 1997. Bayesian phylogenetic inference using DNA sequences: a Markov chain Monte carlo method. *Molecular Biology and Evolution.* 14:717-724.

[125] Wooyoung Jang; Pan, D.Z.; , "Application-Aware NoC Design for Efficient SDRAM Access," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* , vol.30, no.10, pp.1521-1533, Oct. 2011.