NETWORK ON CHIP BASED HARDWARE ACCELERATORS FOR COMPUTATIONAL BIOLOGY

By

SOURADIP SARKAR

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

DECEMBER 2010

To the Faculty of Washington State University:

The members of the committee appointed to examine the dissertation of SOURADIP SARKAR find it satisfactory and recommend that it be accepted.

Partha Pratim Pande, Ph.D., Chair

Ananth Kalyanaraman, Ph.D. Co-chair

Benjamin Belzer, Ph.D.

ACKNOWLEDGEMENT

It is time to pause and reflect back at the end of those months during which I had been working on my Doctoral research. Time it is to thank the people I had been working with.

First and foremost, I would thank my adviser Dr. Partha Pande. It was a pleasure working with him. I am indebted to him for his help during a critical phase of my life. Next, I would thank my co-adviser Dr. Ananth Kalyanaraman. It was a privilege working with him and I would always cherish the joint meeting we used to have. Special thanks go to my colleagues Mr. Turbo Majumdar and Mr. Gaurav Kulkarni for having helped me in the design and implementation of the hardware platform.

I would keep fond memories of the Low Power and Robust Nanosystems Lab, WSU where I worked. The atmosphere in the lab was always cordial. I take this opportunity to thank my colleagues Dr. Amlan Ganguly, Mr. Sujay Deb, Mr. Kevin Chang, and Mr. Chen Cheun Hung.

Lastly, my parents Mr. Dipankar Sarkar and Mrs. Seema Sarkar whose words of encouragement I recall the most.

NETWORK ON CHIP BASED HARDWARE ACCELERATORS FOR COMPUTATIONAL BIOLOGY

Abstract

by Souradip Sarkar, Ph.D.
Washington State University
December 2010

Chair: Partha Pratim Pande

The focus of this thesis is the design and performance evaluation of Network on Chip (NoC) based multi-core hardware accelerators for computational biology. Sequence analysis and phylogenetic reconstruction are the two problems in this domain which have been addressed here. The basic characteristic of sequence analysis is that it is data intensive in nature whereas the kernel operation in phylogenetic reconstruction is compute intensive. Due to exponentially growing sequence databases, computing sequence alignment at a large-scale is becoming expensive. An effective approach to speed up this operation is to integrate a very high number of processing elements in a single chip so that the massive scales of fine-grain parallelism inherent in this application can be exploited efficiently. Network-on-Chip (NoC) is a very efficient method to achieve such large scale integration. The phylogenetic reconstruction application involves solving the breakpoint median problem which reduces to solving multiple instances of the Traveling Salesman Problem (TSP). Specifically, we (i) propose optimized NoC architectures for different sequence alignment algorithms that were originally designed for distributed memory parallel computers, (ii) a custom NoC architecture for solving the breakpoint phylogeny problem (iii) provide a thorough comparative evaluation of their respective performance and energy dissipation. While accelerators using other hardware architectures such as FPGA, General Purpose Graphics Processing Unit (GPU) and the Cell Broadband Engine (CBE) have been previously designed for biocomputing applications, the NoC paradigm enables integration of a much larger number of processing elements on a single chip and also offers a higher degree of flexibility in placing them along the die to suit the underlying algorithm. The results show that our NoC-based implementations can provide above 10^2-10^3-fold speedup over other existing solutions. This is significant because it will drastically reduce the time required to perform the millions of alignment operations that are typical in large-scale bioinformatics projects.

TABLE OF CONTENTS

List of Tables

List of Figures

Dedication

This dissertation is dedicated to all the proponents of Network-on-Chip design philosophy.

Chapter 1

INTRODUCTION

1.1 Hardware Acceleration in Computation Biology

The role of computing in molecular biology research has never been more defining. The interdisciplinary field of computational biology focuses on developing algorithmic techniques and tools for solving biological problems. The essence of the study lies in the development of novel statistical, and computational methods and mathematical models for better understanding of biological systems. Computational genomics involves study of the genomes obtained through genome sequencing techniques of cells. Until a decade ago, only a handful of genome sequences were available and thus simple software implementations yielded acceptable performance. However, due to recent advances in DNA sequencing technologies, sequence data for more than a thousand species are now available in public databases and more large-scale sequencing efforts are currently underway. The key to the recent advances in processing vast amounts of biological data is the interdisciplinary alliance between biologists and computer scientists. Biologists are

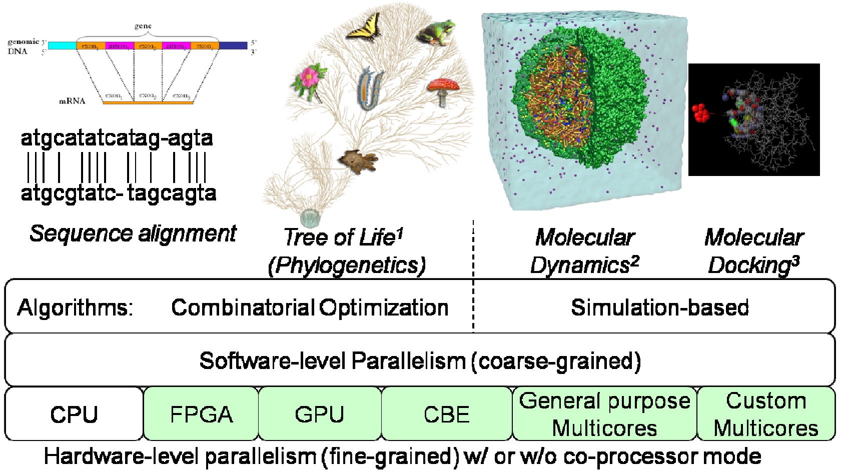

atgcatatcatag-agta
atgcgtatc-tagcagta
Sequence alignment
Tree of Life1 (Phylogenetics)
Molecular Dynamics2
Molecular Docking3
Algorithms:
Combinatorial Optimization
Simulation-based
Software-level Parallelism (coarse-grained)
CPU
FPGA
GPU
CBE
General purpose Multicores
Custom Multicores
Hardware-level parallelism (fine-grained) w/ or w/o co-processor mode

Figure 1.1. Biocomputing applications benefiting from software and hardware acceleration.

responsible for generating the data, and envision the problem, whereas the computer scientists' responsibility is developing efficient algorithms, software suites and hardware solutions for efficient computation. The outcome from these consorted efforts are already benefiting the greater scientific community and opening new venues for new interdisciplinary research. Data processing for biocomputing applications is currently done in software, which often takes a very long time. For instance, aligning even a few hundred sequences using progressive multiple alignment tools consumes several CPU hours on state-of-the-art workstations. Large-scale sequence analysis, often involving up to tens of millions of sequences, has become a mainstay as well as one of the primary bottlenecks in the path to scientific discovery. The biocomputing domain also hosts a set of compute-intensive applications wherein the underlying problems are proven to be computationally intractable (e.g. phylogenetic tree computation, protein folding). These aspects collectively make biocomputing a domain that has the potential to immensely benefit from the incorporation of the latest advancements in circuit design and evolving hardware architectures.

Figure 1.1 shows a high level outline of the current state of computing in the computational biology domain. These problems can broadly be classified into two classes: (i) combinatorial optimization problems and (ii) simulation-based approaches. In this dissertation, we focus on two different classes of computational genomics problems that can benefit from the advances in hardware acceleration techniques. Both the targeted problems belong to the class of combinatorial optimization. One is data intensive in nature while the other is and compute intensive. The problems being (i) pairwise sequence alignment and (ii) phylogenetic reconstruction.

The discovery of biomolecular sequences and exploring their roles, interplay, and common evolutionary history is fundamental to the study of molecular biology. The different sequences that fill complementary roles in the cell are: DNA, RNA and protein. The most predominant compute operation that is carried out in nearly all sequence analysis projects is *pairwise sequence homology detection*, which aims at measuring the similarity, differences or evolutionary relationships between two DNA, RNA or protein ”sequences” (represented as strings over a fixed alphabet). The alignments are broadly classified as *global alignment*, *semi global alignment* and *local alignment*. Global alignment is generally used to compare two protein sequences from a closely related gene family or two homologous genes. The semi global alignment can be used to align fragments of DNA from shotgun DNA reads and create a larger inferred sequence, useful in genome assembly. Local alignment is usually used for finding conserved domains among protein sequences. The most widely used methods are variations of the dynamic programming (DP) algorithm [1] [2] that computes a two-dimensional table, with rows and columns representing the character sequence of the two strings being compared. These methods assign scores to insertions, deletions and replacements, and compute an alignment of two sequences that corresponds to the least costly set of such mutations. Such an alignment may be thought of as minimizing the evolutionary distance or maximizing the similarity between the sequences under consideration. These operations are used almost on a daily basis by molecular biologists, and also in all genome sequencing projects of any scale. While the task of carrying out a single pairwise sequence comparison is in itself computationally light-weight (in milliseconds) on traditional machines, performing the often necessary millions or even billions of such comparisons could easily become prohibitive, without the use of a supercomputer or other specialized hardware [3]. To pace up the computation further, various heuristics attempt to approximate the optimal solution, so that large databases can be searched on commonly available clusters. In these heuristics, the measure of similarity is implicit in the algorithms rather than explicitly defined as a minimal cost set of mutations. Despite all these efforts, the cost of computation still remains. For example, a recent analysis [3] of over 28 million metagenomic sequences took an aggregate of 10^6 CPU hours, a task that took months to complete even after parallelization at the coarse level using a combination of 2,300 processors and high-end memory systems. Our experiments show that running the multiple sequence alignment tool ClustalW [4] even on hundreds of sequences requires several hours on state-of-the-art workstations. Since the amount of biological data is expanding at such a massive rate, there is a compelling need for high performance computing solutions.

Phylogenetics is the study of evolutionary relationships between organisms based on their underlying genetic content. Inferring phylogenetic relationships is important to biologists especially in biomedical research, drug design, and protein structure prediction. Accurate phylogenetic reconstructions involve significant effort due to the difficulties of acquiring the primary biological data and the computational complexity of the underlying optimization problems. Phylogeny is most commonly used for comparative study, where a biological question is answered by comparing how certain biological characters have evolved in different lineages. Some of the fields in which this comparative method finds application is adaptation, development, physiology, gene function, vaccine design, and modes of speciation. The “Tree of Life” is an example of an ambitious project for inferring the phylogeny linking all known life forms. Typical probability models of evolution used for this purpose are Jukes-Cantor (JC) and General Time Reversible (GTR). Unlike sequence alignment, the computational intractability of the problem is the primary stumbling block to advance the state of research in phylogenetic inference, as the underlying problems have been proven to be NP-Hard under various formulations [5]. The choice amongst the main strategies – neighbor-joining, Maximum Parsimony (MP) and Maximum Likelihood (ML), and Markov Chain Monte Carlo (MCMC) - often depends on the nature of the problem at hand. The neighbor-joining methods tend to be polynomial time and reasonably fast from implementation perspective, but they often produce sub-optimal estimates of evolutionary history. The Maximum Parsimony and Maximum Likelihood are difficult optimization problems but they are more preferred among biologists (empirical and simulation results confirm). MP is an NP-hard optimization problem in which the tree with the minimum total number of changes is sought (Hamming Distance Steiner Tree problem); ML is also an NP-hard problem, which is defined in terms of an explicit parametric stochastic model of evolution.

The inherent advantage of ML over MP is the statistical consistency. This implies it is guaranteed to return a correct solution with high probability if the sequences are sufficiently long. However, likelihood analysis is even harder in practice than MP. Both these approaches require substantial amount of time for acceptable levels of accuracy on even moderate sized datasets.

We focus on Maximum Parsimony, which is quite efficiently implemented in software packages like PAUP [6], and GRAPPA [7], and quite effective at producing good MP analyses on fairly big datasets. Heuristics for this problem have been used to construct majority of published phylogenies, and so MP is a major approach to phylogeny estimation.

One of the distance measures of particular relevance to gene rearrangement-based phylogenies is *breakpoint distance*. Given a reference set of n genes $\{g_1,g_2,\ldots g_n\}$, 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 [8]. 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) 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 proved to be NP-Hard [9]. An algorithm called GRAPPA computes an exhaustive search across all possible trees ($\equiv 3*5*7*\ldots*(2N\text{-}5)$ trees), and iteratively runs multiple instances of a TSP solver for scoring each tree. Given the large number of trees to evaluate, phylogenetic reconstruction can easily become heavily compute-intensive – taking days to weeks of compute time – for even a modest number of taxa and genes. More importantly, over 99% of the total run-time gets typically spent in computing TSP instances [10].

Both sequence alignment and phylogenetic reconstruction are challenging because of the following factors:

- **Volume of Data:** High throughout sequencing is rapidly growing the amount of genomics data, thus imposing severe pressure on existing computational infrastructure for processing them. The rate of data processing is lagging behind the rate of generation of data, and this demands faster and more efficient compute solutions.
- **Resource Costs:** Cost of computation is another important issue. As the volume of computation increases, the cost (being a function of the number of computation steps)

also grows. This cost includes memory, power and time. This calls for the need of power efficient and high throughput computation.

- **Variance amongst data:** The data sets in nature have varying degrees of variance. So, the computational techniques should be robust enough in converging to a solution (in a reasonable amount of time), even if not the most optimal.

To address all the aforementioned issues and pace up the data processing time, several hardware accelerators have been proposed recently, of which general purpose homogenous multi-core (dual and quad core Intel and AMD processors), FPGA-based reconfigurable hardware platforms, Graphics Processing Unit (NVIDIA GPUs), and Cell Broadband Engine (CBE) are notable. All the above mentioned systems primarily rely on software and use existing hardware platforms to map algorithms. For large-scale deployment of a data-intensive application, performance and scalability are of major concerns and therefore it is desirable that the hardware implementation is optimized to suit the exact computation and on-chip communication patterns that the application code generates. In this work, we choose to explore performance of Network on Chip (NoC)-based hardware accelerator as it enables integration of exceedingly large computational and storage blocks in the same chip. It is the digital communication backbone which interconnects the components on a multicore System-on-Chip (SoC). Power and wire design constraints are further pushing the adoption of this new paradigm for designing larger multicore chips, which incorporates modularity and explicit parallelism.

1.2 Contributions

The contributions in this dissertation are as follows:

- **Design of NoC Hardware Accelerator for SA**: We focus on developing NoC based hardware accelerators for two different space and time optimal algorithms for performing sequence alignment operations. The complete work involved designing a custom scalable core and the interconnection fabric, which included the on-chip switches/routers. The switch used here is circuit-switched instead of the generic packet switched, as the nature of communication is very regular and happens in a lock-step fashion. The information exchange is also fine grain, as just a single integer needs to be communicated.

- **Design of NoC Hardware Accelerator for MP**: We designed the hardware accelerator platform for solving the TSP problem using branch and bound depth first search (DFS) approach. The entire design consisted of the individual processing engines and the switch design. The communication infrastructure has been designed for two different network topologies, namely flat mesh and quad-tree and they have been compared. In order to model the entire communication network, and communication events occurring in solving a given graph, a Multi-threaded software program was developed. We later used this statistics, for getting an estimate of the total solution time for a particular graph.

  We have successfully demonstrated how our implementation achieved three to four orders of performance gain as compared to the other hardware accelerator platforms.

1.3 Thesis Organization

The dissertation is organized as follows. Chapter 2 describes the related work and an overview on the current hardware accelerators for these applications and their advantages and disadvantages which led us to consider NoC for the target platform. In Chapter 3, we present the first problem, namely sequence alignment where we formulate the problem; describe the corresponding algorithms and the hardware design. Subsequently, we present the experimental results and compare it with the other existing solutions. In Chapter 4, we present our second problem, on phylogenetic reconstruction using maximum parsimony. This is followed by the description of the algorithm, mapping of the same in the hardware and finally we report the results we achieved, in comparison with the serial software solution. Chapter 5 concludes the dissertation with a discussion on future research directions.

1.4 Reference

[1] T.F. Smith and M.S. Waterman, "Identification of common molecular subsequences", Journal of Molecular Biology, 1981, 147: pp. 195-197

[2] R. Ho, K. W. Mai, M.A. Horowitz, “The Future of Wires”, Proceedings of the IEEE, Vol. 89 Issue: 4, April 2001 pp. 490–504.

[3] S. Yooseph et al., “The Sorcerer II Global Ocean Sampling Expedition: Expanding the Universe of Protein Families”, Public Library of Science Biology, 2007, 5(3):e16doi:10.1371/ journal. pbio. 0050016.

[4] J. Thompson et al., “CLUSTALW: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice”, Nucleic Acids Research, 1994, 22: pp. 4673–4680.

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

[6] http://paup.csit.fsu.edu/.

[7] Moret, B.M.E., Wyman, S., Bader, D.A., Warnow, T., and Yan, M., "A new implementation and detailed study of breakpoint analysis," *Proc. 6th Pacific Symp. on Biocomputing (PSB 2001)*, Hawaii, World Scientific Pub., 2001, pp. 583-594.

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

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

[10] J. Bakos and P. Elenis, “A Special-Purpose Architecture for Solving the Breakpoint Median Problem,” IEEE Transactions on Very Large Scale Integration Systems, vol. 16, 2008, pp. 1666-1676.

Chapter 2

Related Work

2.1 Background

Several hardware accelerators have been previously developed for PSA. These accelerators are based on general purpose multicores [1], FPGAs [2], [3], GPUs [4] or Cell Broadband Engine (CBE) [5] [6]. The general purpose multi-core and CBE processors support Multiple-Program Multiple-Data (MPMD) model, while GPU processors support Single-Program Multiple-Data (SPMD) model. In addition, the memory hierarchies for these architectures have distinct characteristics. In the general purpose multi-cores, the cache is entirely handled by hardware, whereas in CBE the system cache is completely handled by software, which implies the programmer has to completely map the data and explicitly load the data prior to its use.

The CBE is a heterogeneous multi-core processor consisting of a general purpose core, which is the Power Core (PPE) and eight other special purpose cores called the Synergistic Processing Elements (SPE). The PPE is responsible for coordinating the execution on the SPEs and run the Operating System. The SPE cores are simple cores and their primary task is execution of a parallel task. Each SPE consists of 256 KB of small private unified memory. The interconnection fabric is called the *Element Interconnect Bus* which is a high bandwidth memory coherent bus facilitating the cores to communicate through DMA data transfers between the local and remote memories. On the contrary the GPU processors consist of a large number of very basic cores and are typically used as accelerators to a host system. GPU is usually connected through a system bus (like the PCI express) to the CPU and is most useful for a certain class of parallel co-processing applications like graphics, signal and image processing. The Compute Unified Device Architecture (CUDA) [7] [8] is a compiler-supported programming model that offers an extended version of the C language for programming recent NVIDIA GPUs [9]. The entire computation is parallelized by executing the same function by N different CUDA threads. The threads accesses data in different levels of the memory hierarchy, and the data organization is crucial for achieving the most efficient implementation. The general purpose multi-core processors include different levels of hardware managed cache, and though the number of cores is smaller compared to CBE and GPU, each core is able to exploit instruction level parallelism (ILP) for efficient single thread execution. The sharing of the internal cache by all cores allows for the efficient data transfer and synchronization between them. For these processors, parallel programming is relatively easy using POSIX Threads (pthreads) or OpenMP directives.

Sequence alignment (SA) may be defined as an order preserving way of mapping characters in one string against characters in the other string or against gap characters. The SA problem is that of aligning two sequences that maximizes the score of aligning. It can be classified into global, semi global and local alignment. Optimal Alignment algorithm for Global Alignment using dynamic programming was proposed by Needleman & Wunch [10]. This involves computing a two dimensional recurrence based table, where the value at each individual cell is dependent on its three neighboring cells. Both the time and space complexity are quadratic in nature. The algorithm for solving the local alignment was given by Smith & Waterman [11], but in essence it is only a slight modification of the Needleman Wunch alignment recursion. Huang [12] proposed a wave front technique for sequence alignment, whereby the cells along each anti-diagonal are computed in each parallel time step. Aluru et al. [13] introduced another technique in which cells along each row can be computed in each time step using the parallel prefix technique. Interestingly, all the above accelerators except for the CBE implementation in [14] use the anti-diagonal based technique for parallelizing the computation of the DP table, as it can be implemented using a simpler layout of processing elements. However, unless the lengths of the two input sequences are approximately equal, the time complexity of the underlying parallel algorithm is sub-optimal. From this perspective, it is imperative to implement and study the effectiveness of the parallel prefix based technique, which guarantees an optimal run time as well.

As for performance results, Weiguo et al. [4] report that the GPU hardware GeForce 7800 GTX can perform up to 700 million DP cells per second, implying an overall time of 1.428 milliseconds for aligning two sequences of length ~1K each. The GPU cores are deployed directly without being optimized to implement the sequence alignment algorithm. The FPGA implementation by Oliver *et al.* [3] reduces the time to 1 millisecond. The prime advantage of FPGA being, it allows logic blocks to be wired together, and the reconfigurable infrastructure which facilitates fast design time. But, the generic reconfigurable architecture also fails to exploit the performance feature completely. The CBE implementation by Sachdeva *et al.* using 16 Synergistic Processing Units (SPUs) runs in 0.65 milliseconds [5]. The other CBE implementation [6], achieves a runtime of ~17 ms using 8 SPUs. Once again these SPUs are not optimized for bioinformatics application suites. As a reference, our own serial implementation of the Smith-Waterman algorithm took 100 milliseconds for aligning two 1K sequences on a 2.3 GHz Xeon CPU. This observation is consistent with the near 100-fold speedups reported by the authors of the aforementioned accelerators.

Hardware accelerators using FPGA have been developed for implementing ClustalW [3], which is a popular multiple MSA program. Since the underlying problem is NP-Hard, ClustalW approximates a solution in polynomial time. A *k*-sequence MSA problem involves computing $\binom{k}{2}$ PSA comparisons. This all-against-all sequence comparison is the dominant phase within

TABLE 2.1. DIFFERENT ALGORITHMS FOR PAIRWISE SEQUENCE ALIGNMENT

Algorithm	Time Complexity	Space Complexity
Aluru's Parallel Prefix	O((m*n)/p)	O(m+n/p)
Huang's Antidiagonal	O($(m+n)^2$/p)	O((m+n)/p)
Rajko and Aluru	O((m*n)/p)	O((m+n)/p)

ClustalW, taking more than 90% of the total time. The FPGA implementation uses Xilinx Virtex II XC2V6000, platform accommodating 92 processing elements (PEs) with a maximum clock speed of 34 MHz. This gives a speed up of around 10 for the overall MSA and about 50 for PSA. It achieves a sustained performance (including all data transfer) of ~1 GCUPS (billion cell updates per second in the DP matrix).

The sequence search tool BLAST (Basic Local Alignment Search Tool) compares nucleotide and protein sequences to sequence databases and calculates the statistical significance of the matches. It proceeds by first identifying a subset of database sequences that have short matching segments with the query sequence and then performing a more thorough evaluation of the query against each short listed candidate. The filtering step is implemented using a look-up-table data structure, and the subsequent evaluation as a unit PSA. Sachdeva et al. [5] implemented BLAST on CBE, consisting of a 64 bit Power Processor Element (PPE), eight Synergistic Processing Elements (SPEs). It achieves a speedup of 2 compared to that implemented on a single Power PC processor. The FPGA BLAST [15] is implemented on Annapolis Microsystems WildstarII-Pro board with two Xilinx Virtex-II FPGAs. The authors have implemented two FPGA BLAST algorithms, namely the TREE BLAST and the SERVER BLAST. The notion behind the former algorithm is that, it can be performed with iterative merging using a tree structure. The FPGA is initialized with the query sequence and scoring matrix. The indexing of the scoring arrays is done using the block RAMs (BRAMs). The database is streamed from the memory to the FPGA. The main component of SERVER BLAST is a systolic array that holds a query string while the database flows through it. This is implemented using a FIFO buffer in FPGA. The performance reported was comparable to that of the dedicated server at NCBI. The GPU implementation of the same problem also returned promising results. Liu et al. [4] demonstrate about 16 fold speedup over OSEARCH, which is a MSA tool. using *nVidia GeForce 7800 GTX GPU*. Mapping of the algorithm onto GPU is done exploiting the fact that all elements in the same anti-diagonal of the DP matrix can be computed independent of each other in parallel. Fragment programs are used to implement the arithmetic operations specified by the recurrence relation. They have reformulated the Smith-Waterman algorithm in terms of computer graphics primitives, in an attempt to exploit the GPU platform for optimum performance.

In phylogenetics research, the primary goal is to reconstruct evolutionary trees that best describe the evolutionary relationship among different species, by observing and characterizing variations at the DNA and protein level. The “Tree of Life” is an example of an ambitious project for inferring the phylogeny linking all known life forms. Typical probability models of evolution used for this purpose are Jukes-Cantor (JC) and General Time Reversible (GTR). Unlike SA, the computational intractability of the problem is the primary stumbling block to advance the state of research in phylogenetic inference, as the underlying problems have been proven to be NP-Hard under various formulations [16].

The following discussion covers different hardware accelerators for MP and ML in the order of increasing problem complexity. Even though it has been shown that parsimony and likelihood give identical results in certain circumstances [17], but empirically biologists claim more accurate results with ML Most of the work addresses ML, which is computationally the most intense of the strategies, involving numerous floating point computations for evaluating the phylogenetic likelihood function (PLF).

Mak and Lam [18] 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 JC 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 [19]. 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 well under a millisecond, based on reported clock speed of 284.152 MHz.

Bakos and Elenis [20] proposed a co-processor design for whole-genome phylogenetic reconstruction using a parallelized version of breakpoint median computation, which is an expensive component of the MP phylogenetic tree inference. The co-processor uses an FPGA-based multi-core implementation of the combinatorial search portion of the Travelling Salesman Problem (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-X interconnects. The best average speedup of 1,005 over software is observed for arithmetic mean computation with 3 cores and 20 lower bound units. The best overall reduction in execution time is by a factor of 417. All these observations are for synthetic data sets and hence difficult to correlate with real-life biological examples.

We focus on MP where the objective problem is to find the tree with the shortest breakpoint length and the leaf nodes labelled by the genomes.   The entire tree construction involves iteratively evaluating the median problem for a 3 leaf tree. That boils down to solving multiple instances of the TSP (as median problem reduces to TSP).

In this work, we are going to show how an NoC-based implementation of PSA, offers significant improvement (in terms of speed) over other hardware accelerators because of its custom made architecture and interconnection topology. The results of deploying such a system achieved two to three orders of magnitude better performance compared to other existing hardware accelerators. NoCs also provide the freedom to design and experiment with different network topologies and their suitability to different algorithmic settings. For Phylogenetic reconstruction, we have developed an NoC hardware accelerator solving the problem using the MP approach, and we had optimized the design further from the perspective of both time and power. The details of the design and the future plan are covered exhaustively in Chapter 4.

2.2 Reference

[1] K. Stavrou, M. Nikolaides, D. Pavlou, S. Arandi, P. Evripidou, P. Trancoso, "TFlux: A Portable Platform for Data-Driven Multithreading on Commodity Multicore Systems," *37th International Conference on Parallel Processing* (ICPP), 2008, pp.25-34.

[2] S. Dydel and P. Bala, ”Large Scale Protein Sequence Alignment Using FPGA Reprogrammable Logic Devices”, Proc. Field-Programmable Logic and its Applications, Lecture Notes in Computer Science, 3203: pp. 23–32.

[3] T. Oliver et al., “Using reconfigurable hardware to accelerate multiple sequence alignment with ClustalW”, Bioinformatics, 2005, 21(16): pp. 3431-3432.

[4] W. Liu, B. Schmidt, G. Voss, W. Mueller-Wittig, "Streaming Algorithms for Biological Sequence Alignment on GPUs", IEEE Transactions on Parallel and Distributed Systems, 2007, Vol. 18, No. 9, pp. 1270-1281.

[5] V. Sachdeva et al., "Exploring the viability of the Cell Broadband Engine for bioinformatics applications " Parallel Computing, 2008 34, 11, pp. 616-626.

[6] A. Sarje and S. Aluru, “Parallel Biological Sequence Alignments on the Cell Broadband Engine”, Proc. IEEE International Parallel and distributed Processing Symposium, 2008.

[7] CUDA Technology, NVIDIA, 2007, http://www.nvidia.com/CUDA.

[8] CUDA Programming Guide 1.1, NVIDIA,2007 http ://developer.download.nvidia. com/ compute/cuda/1_1/NVIDIA_CUDA_Programming_Guide_1.1.pdf

[9] J. Montrym and H. Moreton, ‘‘The GeForce6800,’’ IEEE Micro, vol. 25, no. 2, Mar/Apr. 2005, pp. 41-51.

[10] S.B. Needleman and C.D. Wunsch, “A general method applicable to the search for similarities in the amino acid sequence of two proteins”, Journal of Molecular Biology, 1970, 48: pp 443-453.

[11] T.F. Smith and M.S. Waterman, "Identification of common molecular subsequences", Journal of Molecular Biology, 1981, 147: pp. 195-197.

[12] X. Huang, “A space-efficient parallel sequence comparison algorithm for a message-passing multiprocessor”, International Journal of Parallel Programming, 1989, 18(3): pp.223–239.

[13] S. Aluru et al., “Parallel biological sequence comparison using prefix computations”, Journal of Parallel and Distributed Computing, 2003, 63: pp. 264–272.

[14] A. Sarje and S. Aluru, “Parallel Biological Sequence Alignments on the Cell Broadband Engine”, Proc. IEEE International Parallel and distributed Processing Symposium, 2008.

[15] M. Herbordt et. al., “Single Pass, BLAST-Like, Approximate String Matching on FPGAs”, *Proc. 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2006, pp. 217-226.

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

[17] C. Tuffley and M. Steel, “Links between Maximum Likelihood and Maximum Parsimony under a Simple Model of Site Substitution”, *Bulletin of Mathematical Biology*, 1997, Vol. 59, No. 3, pp. 581-607.

[18] T. S. T. Mak and K. P. Lam, “High Speed GAML-based Phylogenetic Tree Reconstruction Using HW/SW Codesign,” *Proc. Computational Systems Bioinformatics*, 2003.

[19] N. Alachiotis et. al., “Exploring FPGAs for Accelerating the Phylogenetic Likelihood Function,” Proc. IEEE International Symposium on Parallel and Distributed Processing 2009, pp 1-8.

[20] J. D. Bakos and P. E. Elenis, “A Special-Purpose Architecture for Solving the Breakpoint Median Problem,” *IEEE Trans. VLSI Systems*, Dec 2008, Vol. 16, No. 12, pp 1666-1676.

Chapter 3

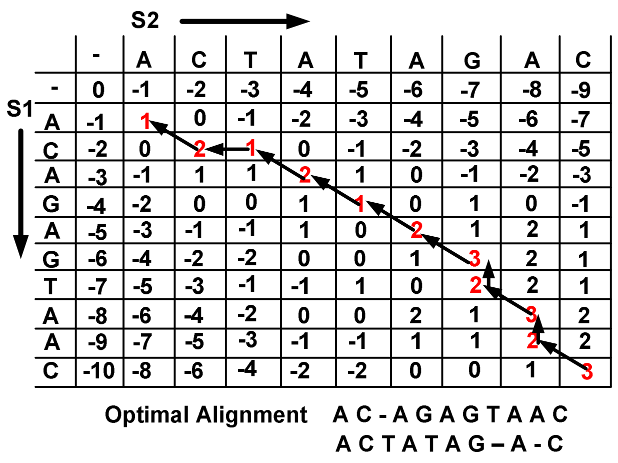
Pairwise Sequence Alignment (PSA)

The focus of this chapter is one of the combinatorial optimization-based bioinformatics problems, namely the sequence alignment. In the following subsections, we will explain the problem, the different algorithms for solving it, and the detailed design and implementation of the NoC-based hardware accelerator. We conclude this chapter with the performance benefits of using such a design approach over existing solutions.

3.1 Algorithms for Sequence Alignment

Sequence alignment is a way of measuring the similarity between two sequences. Algorithmically, comparing two sequences (or strings) is modeled as a combinatorial optimization problem. Characters of one sequence are “aligned” against characters of the other in an order-preserving manner, and only a selected set of operations are permitted at each aligning position: (i) “match” one character with another, (ii) “mismatch” (or substitute) one character with another, and (iii) align one character with an empty (called “gap” and denoted by “-“) symbol on the other string. Through a scoring scheme, a positive score is assigned for similarities (match) and negative scores are assigned for differences (mismatches and gaps). The task of computing an optimal alignment is therefore a task of identifying an alignment with the maximum possible overall score.

Computing an optimal PSA is a well-studied problem [1] [2] [3]. While there are several variants of the problem, the complexities of the underlying Dynamic Programming (DP) algorithms are identical. Given two sequences s_1 and s_2 of lengths m and n respectively, an optimal PSA can be computed sequentially using dynamic programming in $O(mn)$ time and



	-	A	C	T	A	T	A	G	A	C
-	0	-1	-2	-3	-4	-5	-6	-7	-8	-9
A	-1	1	0	-1	-2	-3	-4	-5	-6	-7
C	-2	0	2	1	0	-1	-2	-3	-4	-5
A	-3	-1	1	1	2	1	0	-1	-2	-3
G	-4	-2	0	0	1	1	0	1	0	-1
A	-5	-3	-1	-1	1	0	2	1	2	1
G	-6	-4	-2	-2	0	0	1	3	2	1
T	-7	-5	-3	-1	-1	1	0	2	2	1
A	-8	-6	-4	-2	0	0	2	1	3	2
A	-9	-7	-5	-3	-1	-1	1	1	2	2
C	-10	-8	-6	-4	-2	-2	0	0	1	3

Figure. 3.1. Example of computing the global alignment between two sequences using Needleman & Wunsch algorithm [3]. The arrows show the optimal path. The following scoring scheme was used: matchscore=1, mismatch penalty=1, gap penalty=1.

$O(m+n)$ space. This is achieved by computing a $(m+1)\times(n+1)$-sized table T, such that T[i,j] contains the optimal score for aligning the prefixes s_1[1..i] against s_2[1..j]. For example, the global alignment[1] score of aligning prefixes s_1[1..i] and s_2[1..j] is given by:

$$T[i,j] = \max\begin{cases} T[i-1,j-1]+\sigma(s_1[i],s_2[j]) \\ T[i-1,j]-g \\ T[i,j-1]-g \end{cases} \qquad (1)$$

where g>0 corresponds to the gap penalty and $\sigma()$ to the score for substituting s_1[i] with s_2[j] or vice versa. As can be noted, the value at T[i,j] depends on the cells T[i-1,j-1], T[i-1,j], and T[i,j-1]. Sequentially, this dependency constraint can be met during computation through a "forward pass" of the table in which the table is computed one row at a time starting from the first row, and within each row computing column by column starting from the first column. At the end of the forward pass, the optimal score is available at T[m,n]. The next step is a "backward pass" in

1

which an optimal path (or equivalently, an optimal alignment) that led to the optimal score is retraced from T[m,n] to T[0,0]. Figure 3.1 illustrates an example DP table along with its optimal path.

3.1.1. Parallelization: There are two main challenges in parallelizing DP algorithms for PSA: i) meeting the dependency constraint without affecting the parallel speedup during forward pass; and ii) computing the optimal retrace without storing the entire DP table during forward pass. To meet these challenges, several coarse-grain parallel algorithms have been previously developed [4] [5] [6] [7] [8]. These algorithms offer varying degrees of computational complexities and ease of implementation. The algorithm by Huang [6] develops on ideas proposed by Edmiston [5] by using the anti-diagonal approach during forward and backward passes. The guiding observation is that the cells along the same anti-diagonal of the DP table are not interdependent and therefore can be computed in parallel. If p denotes the number of processors, then this algorithm requires $O\left(\frac{(m+n)^2}{p}\right)$ time and $O\left(\frac{m+n}{p}\right)$ space. Aluru et al. [4] devised an alternative strategy that overcomes the dependency constraint by reformulating the problem of computing the scores within a row in parallel using the parallel prefix operation. This algorithm requires $O\left(\frac{m \times n}{p}\right)$ time and $O\left(m+\frac{n}{p}\right)$ space, assuming m= $O(n)$. The algorithm by Rajko and Aluru [25] uses a combination of these ideas to arrive at a more complex albeit time- and space-optimal solution – i.e., $O\left(\frac{m \times n}{p}\right)$ time and $O\left(\frac{m+n}{p}\right)$ space.

For mapping onto the NoC architecture, we based our choice on the following factors: i) parallel run time and space complexities, (ii) relative ease of adoption to the on-chip framework, and (iii) the potential to fully benefit from the on-chip communication network. Based on these factors, we selected the algorithms by Aluru et al. [4] and Huang [6] for implementations in this work. While it will be ideal to also evaluate the optimal algorithm by Rajko and Aluru [7], it is highly complex for implementation. It is to be noted that none of the previously proposed hardware accelerators implemented the parallel prefix approach.

In what follows, we briefly outline the main ideas behind these two algorithms. For convenience, we will refer to the algorithm by Aluru et al. as the “PP algorithm” (for parallel prefix), and the algorithm by Huang as the “AD algorithm” (for anti-diagonal).

We implemented three of the most popular variants of the alignment problem – global [2], local [1], and semi-global [9]. To best reflect practical application, we implemented the affine gap penalty function model [3], in which the gap penalty function grows linearly with the length of the gap in an alignment. Algorithmically, this is achieved by computing three DP tables (T_1, T_2 and T_3) instead of one DP table. However, the underlying run-time and memory complexities for computing alignments based on the affine gap model are exactly the same as that of the single-table constant gap model. The actual time and memory costs in practice are expected to only increase by a factor of 3. Because of this algorithmic equivalence and for ease of exposition, we will describe the parallel algorithms below for the single-table constant gap model, even though we implemented the more generic affine gap model.

3.1.2. The PP approach: The PP algorithm partitions the input sequence s_2 into p pieces such that each PE receives roughly n/p characters (as shown in Figure 3.2). The other input sequence s_1 is made available to all the PEs one character at a time. Throughout, we will assume p to be a power of 2 although the algorithm can be easily extended to arbitrary processor sizes through a virtual padding scheme. The $(m+1)\times(n+1)$-sized table T computation is divided into p roughly equal parts, such that PE p_i is assigned the responsibility of computing the i^{th} block of

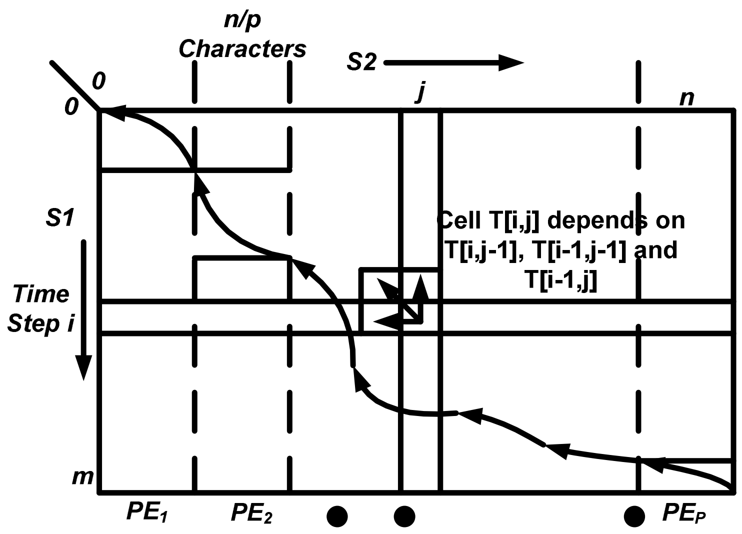


Figure. 3.2. Computation of the DP table in parallel using p processors in the parallel prefix approach.

$O(n/p)$ columns. The forward pass in table computation proceeds iteratively, row by row, such that at any given iteration i, all the PEs participate in computing row i in parallel. We identify each iterative step as one “time step”. Within each row, the algorithm reduces the problem of computing the recurrence in (1) to the problem of computing the following recurrence

$$X[j] = \begin{Bmatrix} w[j] + jg \\ X[j-1] \end{Bmatrix} \qquad (2),$$

where $0 \le j \le n$ and w[j] is obtained by local computation (without any need for

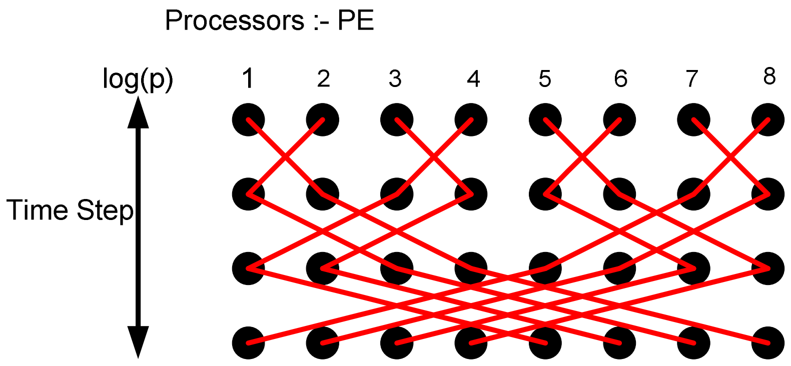


Figure. 3.3. Communication pattern generated by the PP algorithm.

communication). Computing this recurrence is equivalent to the problem of finding n+1 prefix maximums, which can be easily accomplished using the PP operation as follows: Since each row is partitioned into roughly $O(n/p)$ blocks and assigned to p PEs, the prefix maximums can be computed by first computing p local maximums in an $O(n/p)$ computation step, and then using $O(\log p)$ communication steps to update their local prefix maximums into global prefix maximums. More specifically, at communication step k (for $0 \le k < (\log p)$), PE_i exchanges its most-recent global maximum with PE_j such that $j=i+2^k$ and the k^{th} least significant bits in the binary representations of i and j are 0 and 1 respectively. For example, in a 4 processor system and at k=0, PE_0 exchanges with PE_1, and PE_2 exchanges with PE_3; at k=1, PE_0 exchanges with PE_2, and PE_1 exchanges with PE_3. The time steps of the inter-PE communication scenario for 8 PEs is shown in Figure 3.3. Consequently, each time step can be completed by performing: (i) an $O(n/p)$ local computation, and (ii) an $O(\log p)$ parallel prefix communication. After the last row is fully computed, the PEs reverse their computation by progressing from last row to top row and retrace an optimal alignment path that yielded the optimal score at cell T[m,n]. However, this would require that the entire table be stored, implying an $O(mn)$ aggregate space complexity. To allow retracing an optimal alignment in just $O\left(\frac{m+n}{p}\right)$ space, each PE stores all the entries in the last column of its block of n/p columns, and then uses this information to retrace. This step can be achieved in $O(n/p)$ computation time and $O(p)$ communication time. In the interest of space, we omit the details of analysis and proofs, and refer the reader to the original paper [4]. It has been proved that the algorithm has an overall computation complexity of $O(mn/p)$ and a communication complexity of $O(m\log p)$ on a distributed memory parallel computer [4].

3.1.3. The AD algorithm: This requires that both the input sequences s_1 and s_2 are made available to all the PEs. For the forward pass, the algorithm proceeds iteratively by computing one anti-diagonal of DP table at each "time step", where an "anti-diagonal" is defined as the subset of cells [i,j] which have the same i+j value. For an $(m+1)\times(n+1)$ DP table, there are a total of m+n+1 anti-diagonals with values 0, 1, 2,.., m+n, and the algorithm computes the tth anti-diagonal at time step t (as shown in Figure 3.4). All the cells within an anti-diagonal can be computed in parallel because the value at any T[i,j] depends on the values already computed and available from previous two time steps. The next question is to identify the PE that will work on each cell of an anti-diagonal. It turns out that the assignment of PEs to cells does not matter for the overall complexity as long as the number of PEs working on an anti-diagonal is maximized. Therefore, in this work, we adopt two different strategies, for the different cell computations by the processors. First that will assign PE_k to the cells in the anti-diagonal that have the form [i,j]

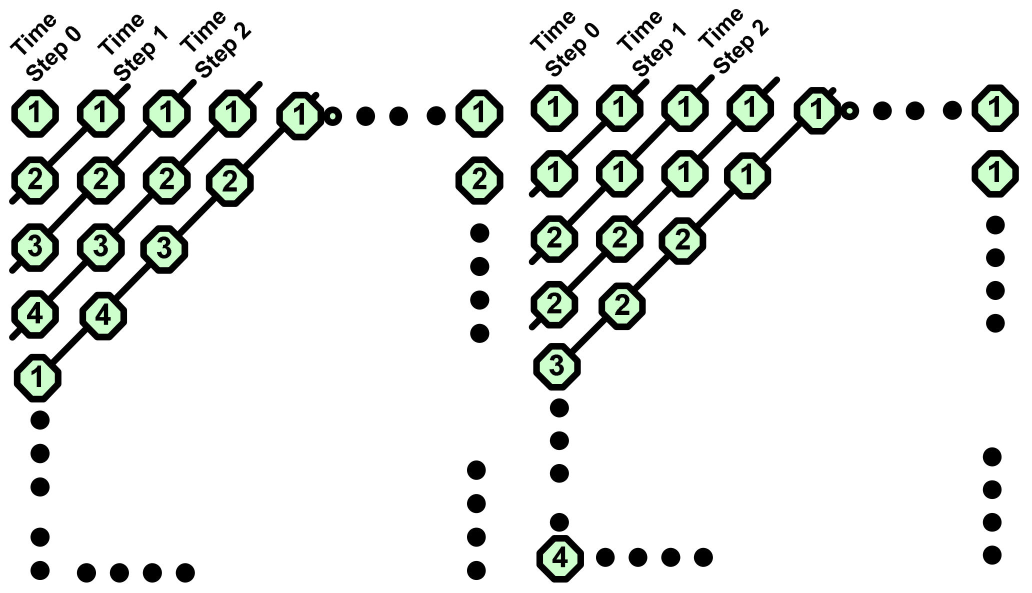


Figure. 3.4. Antidiagonal table computation. (a) Strategy 1, (b) Strategy 2. The numbers within the cells represents the PE responsible for computing it.

such that (i mod p) = k. This is shown in Figure 3.4 (a). The second strategy is shown in Fig 3.4 (b), where the interprocessor communication is further reduced. This way of processor allocation also results in reduction of the computation time, as the internal cells do not have to wait for the communication results from neighbouring cells. This significantly improves the amortized total time, even though the participation from all the processors is not uniform at different instances (in comparison with the previous strategy). We selected this cyclic allocation scheme because the communication pattern that emerges from such a setting is a simple neighbourhood communication – i.e., the data that PE_k needs to compute a cell (i,j) are present either within itself or in PE_{k-1} (as shown in Figure 3.4).

For the backward pass, the main challenge is to reconstruct an optimal path in the absence of the entire DP table stored at the end of the forward pass, as otherwise the space complexity will be Θ(mn). The algorithm by Huang uses a variation of the Hirschberg technique for space reduction [10], by identifying the cell (i',j') in the middle anti-diagonal through which an optimal path must have passed. Once such a cell is found, the problem space can be recursively subdivided into reconstructing the paths on the left top and right bottom sides of (i',j'). In this work we developed another variant that directly uses the Hirschberg technique. In this scheme, the special cell (i',j') is defined to be $(\lceil m/2 \rceil, j)$ through which an optimal path is guaranteed to pass. We achieve this by propagating all possible "candidate" cells during the forward pass such that the candidate that propagates to the last cell (m+1)×(n+1) is the winner. Once such an (i',j') is found, the original problem of retracing the DP table from (m+1,n+1) back to (0,0) reduces to two disjoint sub-problems: i) retrace from (m+1,n+1) back to $(\lceil m/2 \rceil, j)$; and ii) retrace from $(\lceil m/2 \rceil, j)$ back to (0,0). These two sub-problems can be solved using recursion. To maintain parallel efficiency during these recursive steps, we partition the processor space into

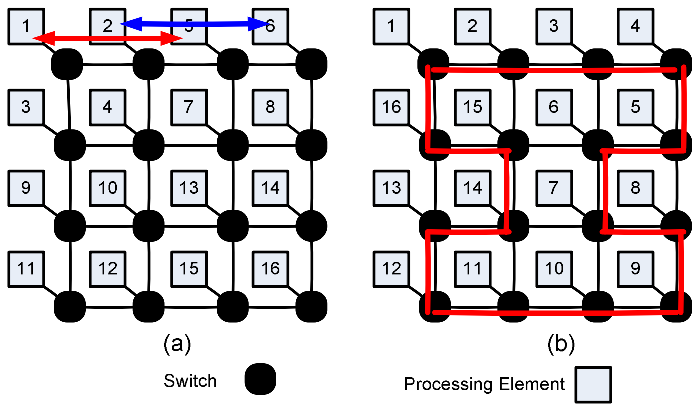


Figure. 3.5(a) Mapping of the PP algorithm into a Mesh. (b) Mapping of the AD algorithm into a Mesh with an embedded ring.

two subsets such that the number of PEs in each subset is proportional to the number of cells for computation in the corresponding recursive step. The AD approach also requires $O\left(\frac{(m+n)^2}{p}\right)$ time and $O\left(\frac{m+n}{p}\right)$ space.

3.2. NoC Implementation

3.2.1 Mapping the Sequence Alignment Algorithms to NoC

Instead of depending on FPGAs or any other existing processing platforms, we designed tiny PEs operating on a particular segment of the table T as explained in Section 3, and integrating them using an NoC.

(a) The PP implementation: The communication is always point-to-point and the PEs are required to exchange a single integer number among them. As an example, the inter-processor communication steps for a system with 8 PEs are shown in Figure 3.3. Consequently, instead of building a full blown packet switched network, a simpler circuit-switched NoC is designed. The total time required to complete a sequence alignment operation depends on the computation time taken by each PE and the communication time needed to exchange data among the PEs. The algorithm is structured such that at every time step, there is a $O(n/p)$ local computation phase followed by a communication phase. The communication carries out a parallel prefix operation in $O(\log p)$ stages, but the overall communication time will depend on the architecture of the NoC and placement of the PEs. Therefore, it is important to place the PEs in the NoC in such a way so as to reduce both the latency and energy dissipation in communication. While there are multiple NoC architectures [11], the hypercubic topology is best suited for the parallel prefix operation. But a physical realization of the NoC is limited by the layout dimension of the chip, which is predominantly 2D in practice. For a system size of p, if we construct a $\log_2 p$-dimensional hypercube then the number of hops between any two PE is always 1. But as shown in Figure 3.5(a), if we embed a $\log_2 p$-dimensional hypercube into a D-dimensional mesh, which is more realistic from an implementation perspective, then the number of hops in the i th communication step is given by (3).

$$L_i = 2^{\left\lfloor \frac{i-1}{D} \right\rfloor} \qquad (3),$$

where i can range from 1 to $\log_2 p$. Hence, the maximum communication latency (in number of hops) between any two PEs is given by (4)

$$L_{\max} = 2^{\left\lfloor \frac{\log_2 p - 1}{D} \right\rfloor} \qquad (4)$$

Given that there are exactly $\lceil \log_2 p \rceil$ time steps within one parallel prefix operation, the total communication time T (in hops) of the parallel prefix operation is given by (5).

$$T = \sum_{i=1}^{\log_2 p} 2^{\left\lfloor \frac{i-1}{D} \right\rfloor} \qquad (5)$$

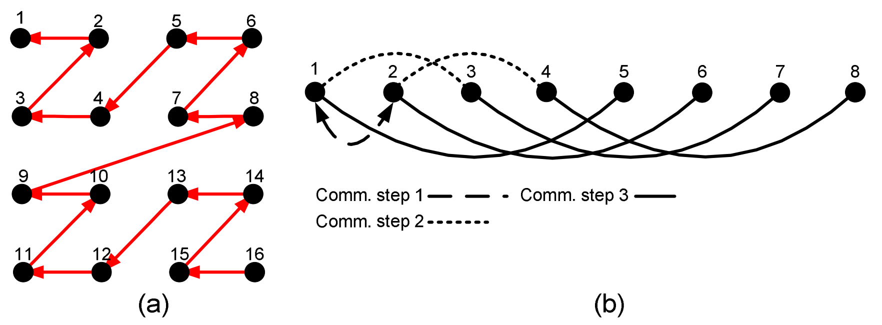


Figure. 3.6. (a) The communication pattern for backward pass in PP (b) The communication pattern for backward pass in AD.

For 2D, the above series and hence T evaluate to $(2\sqrt{2} \times p - 2)$ if log p is even, and $((\frac{3}{\sqrt{2}})p - 2)$ otherwise. Given that there are $O(\log p)$ time steps, the above results yield an average of $O(p/\log p)$ number of hops per time step for both the cases.

It is not practical to implement any arbitrarily higher dimensional hypercube. For example, a system with 64 PEs would necessitate construction of a 6D hypercube. Therefore, for investigation, we designed a 2D mesh-based NoC for carrying out sequence alignment. When the communication pattern shown in Figure 3.3 is mapped to a 2D mesh-based NoC, even a data exchange between two hypercubic neighbors may cost several hops. But a well-defined property of the communication pattern in parallel prefix algorithm is that PEs do not communicate arbitrarily. As an example, in a system with 16 PEs, if the placement of Figure 3.5(a) is followed then the worst case communication latency arises while communicating between PEs separated by two hops. With increasing system size this worst case communication latency will be more. For a system with 64 PEs the worst case latency in a communication step will be four hops.

As explained above, the forward pass is followed by a backward pass operation. This step is implemented using p-1 neighbor PE communication exchanges, as the PEs regenerate the path from cell [m,n] to [0,0]. We modeled this communication pattern as a Z space filling curve as shown in Figure 3.6(a).

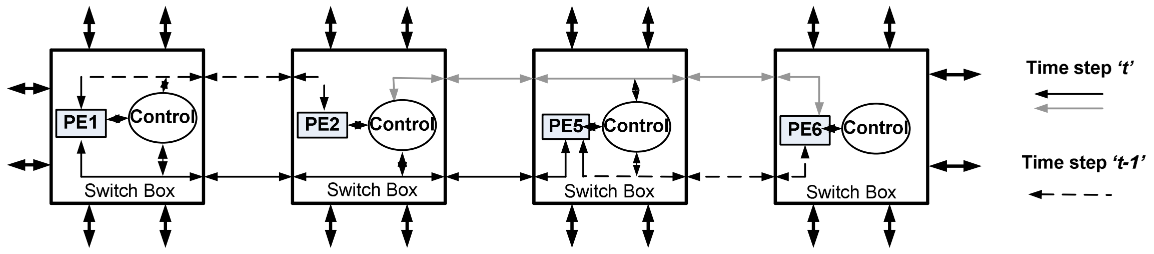


Figure. 3.7. Establishment of communication links facilitating Bypass during parallel prefix.

(b) The AD implementation: In the AD approach, the forward pass requires only a neighborhood PE communication pattern, as explained in Section 3. More specifically, the three values required to compute T[i,j] are available from the previous two anti-diagonals, and because of the cyclic allocation strategy that we used to map PEs onto the anti-diagonals (see Figure 3.4), these cells are either present in the same PE or the previous PE. The exception is the first PE which will depend on the last PE due to the cyclic allocation. Therefore, it suffices to use a ring topology. In our implementation, we achieve this by embedding a ring into the mesh, and following the Moore space filing curve, which is similar to the Hilbert curve [12] for PE numbering. The placement of PEs is shown in Figure 3.5(b).

This interconnection enables single cycle communication among the neighboring nodes. At every time step, each PE works on one anti-diagonal of the DP table. If the length of an anti-diagonal is greater than the number of PEs, then the cells are computed in multiple stages. The number of such stages is given by:

$$Stages\ per\ Anti\ Diagonals = \frac{Average\ Anti-Diagonal\ length}{\#\ PEs} \approx \frac{m}{2p} \quad (6),$$

where $m \leq n$ without loss of generality. The communication steps follow each of the computation steps at each cell. This implies that the total number of data exchanges is proportional to $(m+1)\times(n+1)$. Note that this result is unlike the PP approach; the

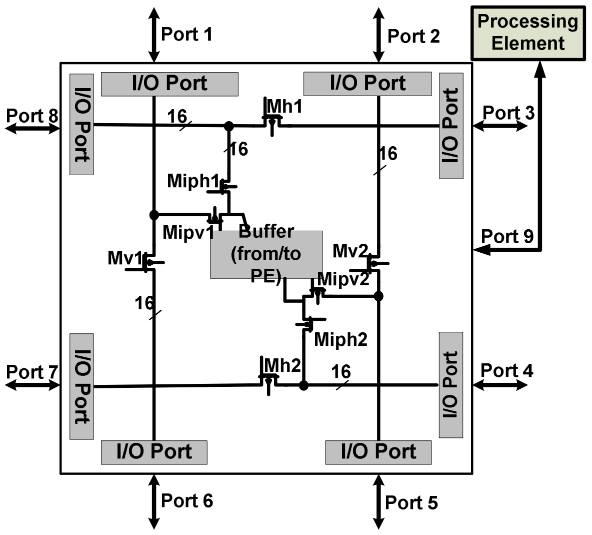


Figure. 3.8. Generic switch architecture for both PP and AD approaches.

communication volume is independent of the number of PEs during the forward pass.

In our backward pass implementation we partition the processors into two sub-groups and the number of processors in each of the sub-groups depends on the number of cells in the two partitions. The processor grouping requires a broadcast operation as shown in Figure 3.6(b) to propagate the new partitioning cell-coordinate to all the PEs, which takes $O(\log p)$ time (where p is the number of PEs).

3.2.2 NoC Switch Design

Due to the deterministic pattern of communication in case of both the PP and AD techniques, we designed simple pass transistor-based switch boxes [13] to forward the data from one PE to the other, instead of designing network routers for data communication.

In the PP approach, data exchanges between two non-adjacent hypercubic neighboring PEs give rise to higher communication delay. To reduce the delay, instead of building a multi-hop or pipelined communication link between two non-adjacent PEs, the switch boxes are designed to establish a direct communication path (unpipelined or single-hop) between the PEs [33]. As an example, for the mesh shown in Figure 3.5(a), a particular communication step requires that PE_1 communicates with PE5 and simultaneously PE_2 communicates with PE_6. In this situation the switchbox connected to 2 should be conFigureured in such a way that a direct communication link is established between 1 and 5 as shown in Figure 3.7. The architecture of a switch is shown in Figure 3.8. Commensurate with all the time steps in the parallel prefix operation (shown in Figure 3.3) the switch is designed to establish direct path between any two communicating neighbors in the vertical and horizontal directions. For a system with 16 PEs the communication steps within the parallel prefix operation are shown in Figure 3.9. In 'Time Step 1' of parallel prefix, the neighboring processing elements communicate (like 1-2, 3-4, 5-6 etc.). For example, to exchange data between PE_1 and PE_2, the following pass-transistors will be on: Miph1 and Mh1 of the switch connected to PE_1, and Miph1 of the switch connected to PE_2. The same switch set-up facilitates data transfer amongst PE_3-PE_4, PE_5-PE_6…PE_{15}-PE_{16}. In the subsequent time steps, other switches involved in the communication are configured accordingly (by turning on suitable pass transistors in the switches). With increasing system size the number of ports in the switches needs to increase to facilitate single-hop or unpipelined communication. On the contrary, the number of ports in the multi-hop scenario does not increase as the message ripples through the intermediate switches. During the backward phase, the data transfer is serial and hence no new modification is required in the existing communication infrastructure.

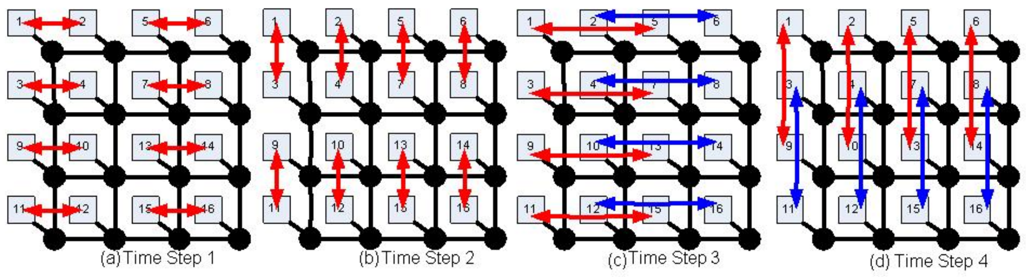


Figure. 3.9. Different time steps in communication during parallel prefix (PP). The shaded arrows represent the different simultaneous communications taking place for each parallel prefix step.

For the AD technique, during the forward phase only neighbouring PEs communicate simultaneously. The backward pass requires broadcasting of information, which is implemented as shown in Figure 3.6 (b). To achieve the simultaneous communication amongst multiple non-adjacent PEs during this phase, the bypass strategy of the PP implementation is adopted here too.

3.3 Experimental Results

3.3.1. Input data: Given that the complexities of the PSA algorithms discussed above depend only on the lengths of the sequences being compared (and not on the sequences content), we used two arbitrary DNA sequences with lengths 1024 characters each in all our experiments. In practice, the length 1024 represents the length range for sequences that can be experimentally generated (or “sequenced”) using a traditional Sanger sequencer [26]. These sequences constitute a typical input in genome sequencing projects, where a massive number of pair-wise alignments are computed over millions of such sequences. However, we also note that there are DNA sequences of a vast length ranges in public databases – from tens to hundreds of characters (e.g., short reads from new generation sequencing), to thousands of characters (genes), to tens of thousands to millions and even billions of characters (fully assembled whole genomes). Even though we selected 1024 for our input tests, our NoC implementations can be used to any of these length ranges as long as the sequences fit in an on-chip memory. Fixing the input size in all the experiments allowed us to conduct a fair comparative evaluation of the different NoC

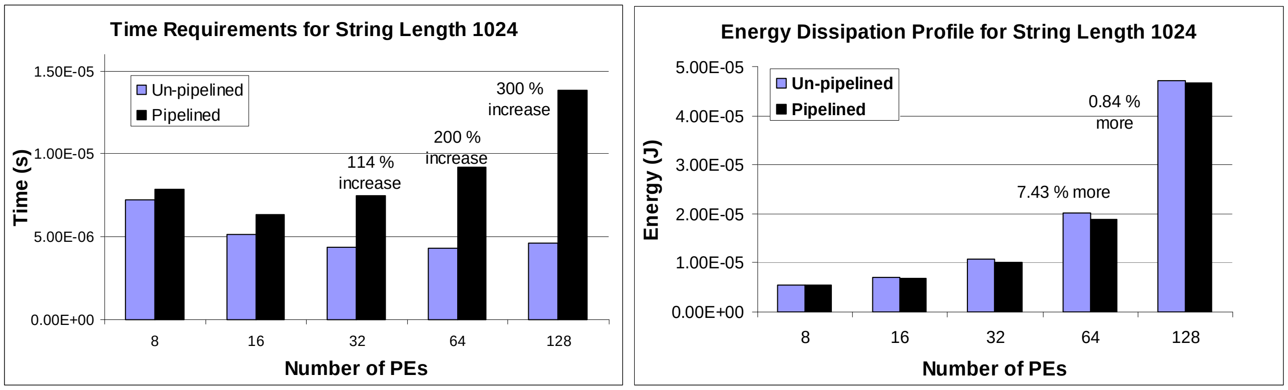


Figure. 3.10. (a) Time requirements comparison for PP (b) Energy dissipation profiles for PP.

architectural topologies in our implementations.

3.3.2. Experimental setup: We present here experimental results of our NoC implementations for the two algorithms described in Section 3: PP and AD. We studied the timing requirements and the energy dissipation for both the cases. Each character of a string was represented using 3 bits. This is because the alphabet size of DNA sequences typically used in practice is 5 ({a,c,g,t,n}). In all our implementations, the PEs need to exchange only integer data among them. Each integer number used during the communication was represented by 16 bits. Each PE was designed to perform the required character comparison, which is the primary unit operation in string alignment. The PEs were implemented in RTL and then synthesized using the 90 nm standard cell library from CMP (http://cmp.imag.fr/). The PEs communicate with each other with the help of switches. The switches mainly consist of pass transistor logic and were designed using Cadence Spectra tools. We considered two types of NoC implementations: (i) a "Pipelined" communication scheme, where all the non-adjacent PEs communicate in multiple hops step by step; and (ii) an "Unpipelined" communication scheme, where the non-adjacent hypercubic neighboring PEs communicate in a single hop with the help of bypass. Performances of both the schemes were compared in terms of energy and timing. The energy dissipation and the total time required in the sequence alignment operation depend on the PEs and the communication infrastructure. The energy dissipation and delay of the communication infrastructure in turn depend on two components, the switch blocks and the inter-switch wires. The energy dissipation and delay of the switch blocks was determined using the CADENCE Spectra tool. The clock frequency of opertation was 1.667GHz. The delay and energy dissipation of the inter-switch wires depend on their capacitance, which was calculated by taking into account each inter-switch wire's specific layout by the following expression

$$C_{\text{interswitch}} = C_{wire} \cdot w_{i+1,i} + n \cdot m \cdot \left(C_G + C_J\right) \quad (7)$$

where C_{wire} represents the capacitance per unit length of the wire, $w_{i+1,i}$ is the wire length between two communicating switches, n is the number of repeaters, m represents the size of those repeaters with respect to minimum-size devices, and lastly, C_G and C_J represent the gate and junction capacitance, respectively, of a minimum size inverter. The energy dissipation and delay incurred by each PE are obtained using Synopsys Design Vision.

3.3.3. Pipelined vs. Unpipelined Implementation

As a first step we compare and contrast the performance of the NoC for the pipelined and the unpipelined implementations. We conduct this analysis only for the PP implementation. For the AD implementation, only neighborhood PEs communicates (in a ring topology) and therefore, all data transfers will be inherently single-hop, no bypass strategy is needed.

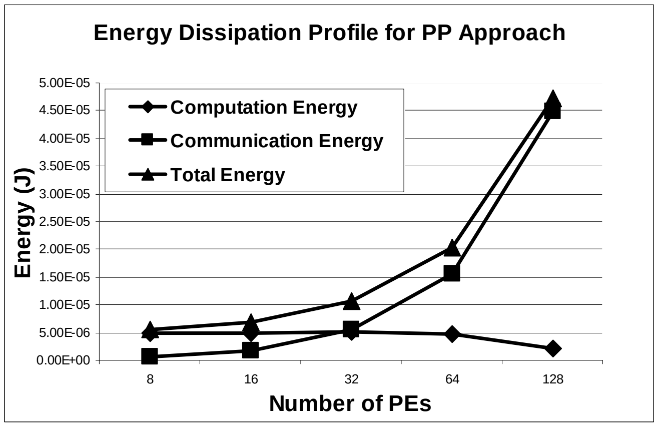
Figures. 3.10 (a) and (b) show the timing and energy dissipation profile of the NoC with varying system size. As can be expected from Figure 3.10(a), the pipelined implementation takes much longer time to complete than the unpipelined implementation. The total time has two parts: communication time and computation time. With increase in system size, the communication time increases because there will be more number of time steps within each parallel prefix operation in both the multi-hop and single-hop scenarios. However, the computation time of each

PE decreases due to reduction in the substring length. But in the pipeline scenario the communication time dominates over the computation time significantly (as an example, 14:1 ratio for a system size of 64). This is true for the unpipelined case as well, but the ratio is smaller (6:1 for the same system). As a result, in the pipelined case the rate of increase of overall time with increasing system size is much higher than the corresponding unpipelined implementation. Contrary to the timing characteristics, the unpipelined case dissipates more energy (as shown in Figure 3.10b). This can be attributed to various factors: (i) the increase in the interconnect length traversed in one communication cycle, (ii) buffers along the path, and (iii) increase in switch complexity.

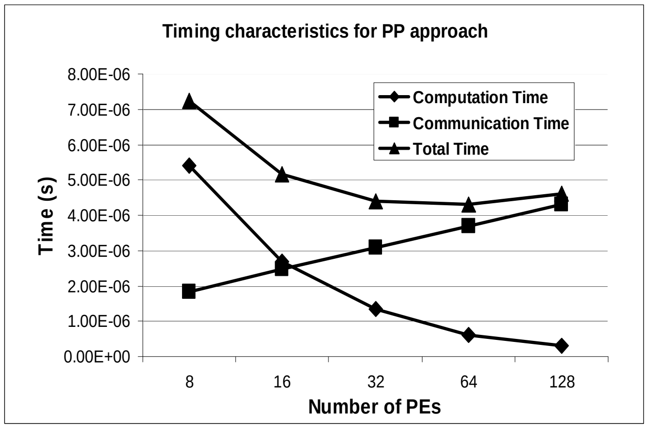
It can be observed from Figure 3.10, that the unpipelined implementation provides significantly more savings in time compared to the pipelined one, while resulting in very little penalty in energy consumption for all system size. As an example, for a system size of 128 PEs the unpipelined implementation achieves more than 300% improvement in time while consuming only 0.84% more energy. This result indicates the value added by the bypass strategy. As a result of this analysis, we adopted the unpipelined strategy in the final implementation of PP, and all the corresponding results presented henceforth are for the unpipelined implementation.

3.3.4. Energy and Timing Characteristics of the PP Approach

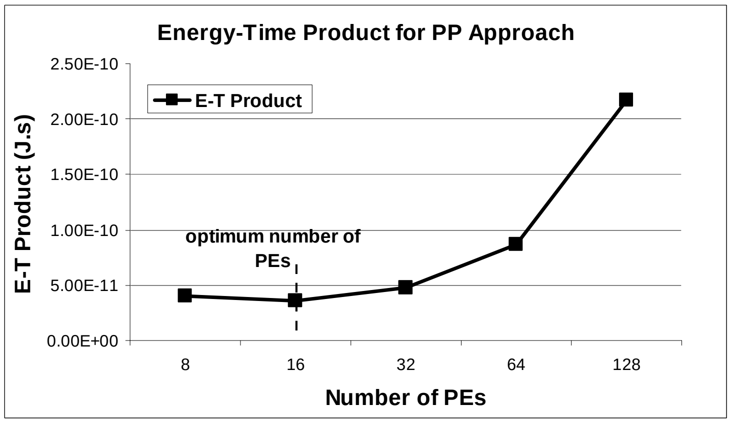
Figure 3.11 shows the energy and timing characteristics of our NoC implementation of the PP approach. Figure 3.11(a) shows the energy dissipation profile with varying the number of PEs for the PP operation. The two contributing factors are the communication and computation energy. The increase in the communication energy with system size is attributed to the increase in total number of communication steps. On the contrary, the computation energy reduces very slowly. With doubling the system size, the work performed by each PE reduces by half. Hence the energy per PE also reduces. It is observed that with doubling system size (halving the string length handled by each PE) the factor of energy reduction per PE is slightly more than two.



(a)



(b)



(c)

Figure. 3.11. (a) System energy profile (b) Timing requirements (c) The E-T product for PP approach.

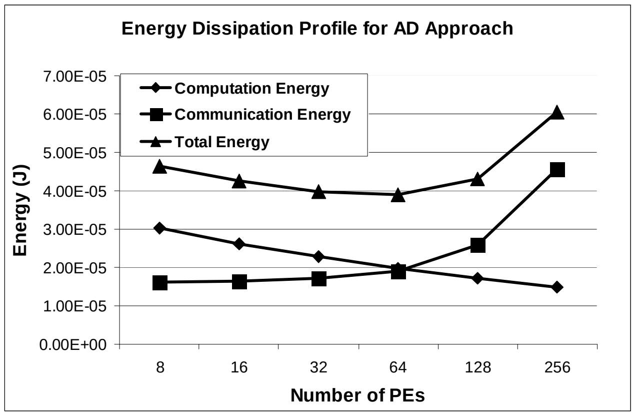
Consequently the total computation energy has a slow decreasing trend with doubling system size. Overall, the net energy trend is dominated by the communication energy.

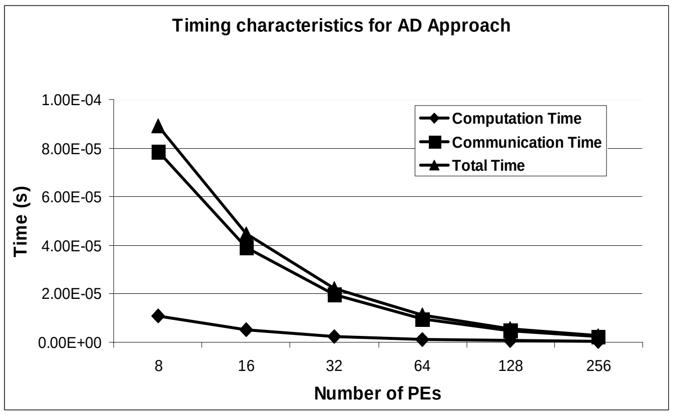
The timing characteristics of our PP implementation are shown in Figure 3.11 (b). For a system with p PEs, each parallel prefix operation involves $O(\log p)$ communication steps. As there are m rows, where m denotes the length of string s_1, the total communication time for the entire alignment operation is $O(m\log p)$. Consequently with increasing number of PEs, the communication time increases. At the same time with increasing system size, the number of columns in the DP table, and hence the overall workload handled by each PE decreases. In fact, the computation time of each PE almost halves with doubling the system size until the input size becomes too small for the system size. This explains the observed trend of the computation time in Figure 3.11 (b). Consequently the total time needed to perform the alignment operation, which is the sum of the computation and the communication time first decreases and reaches a valley, but beyond a certain number of PEs, it again starts increasing. In all our experiments, we observed that more than 90% of the overall time was spent on the forward pass of the algorithm.

To determine the optimum number of PEs we consider the variation of the energy-time product with respect to system size. From Figure 3.11(c) it can be observed that for comparing two strings of length 1024, for the PP algorithm the optimum number of PEs turns out to be 16.

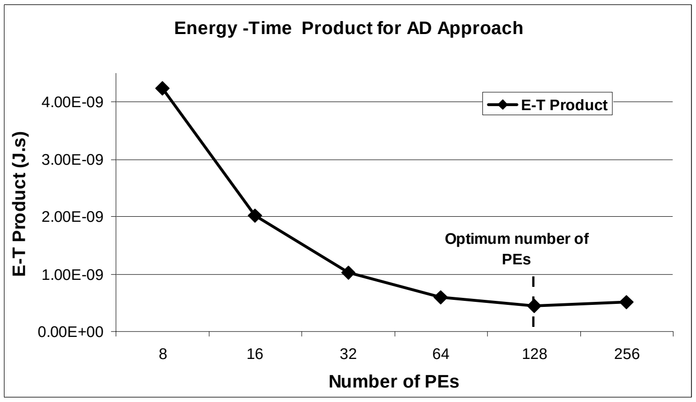
3.3.5. Energy and Timing Characteristics of the AD Approach

Figure 3.12 shows the energy and timing characteristics of our NoC implementation of the AD approach using the fist strategy (refer section on AD algorithm). Figure 3.12(a) shows the energy characteristics. In the first case of AD approach, the total number of communication steps during forward pass is *O(mn)*, irrespective of the system size, where m and n are the lengths of the two strings. This is because there is a data exchange at every cell of the DP table. Consequently the communication energy in the forward pass remains unchanged with increase in the system size. However, during the backward pass, the communication energy increases with the system size due to the broadcast operation. This explains the slow rise in the total communication energy shown in Figure 3.12(a).





(a) (b)



(c)

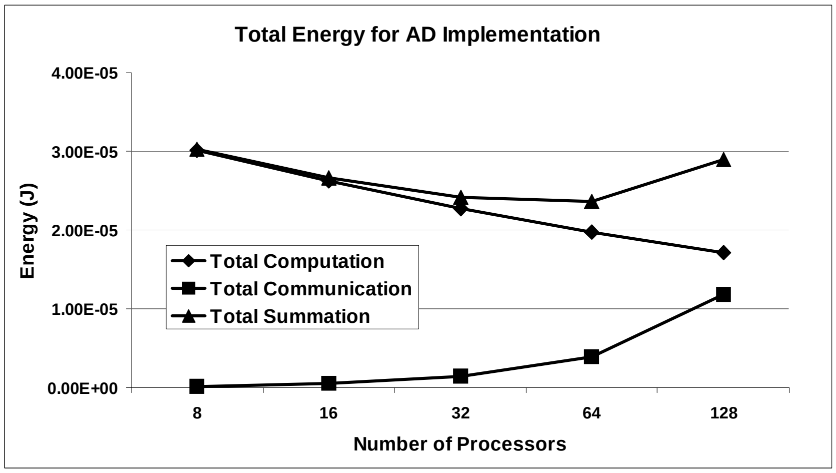
Figure 3.12. (a) System energy profile (b) Timing requirements (c) The E-T product plot for AD approach.

The computation energy is given by:

$$Comp.\,Energy_{AD} = E_p \times PE \times Comp.\,cycles \qquad (8)$$

where E_p is the energy of a single processor per computation cycle. As the system size doubles, the number of total computation cycles per processor halves. Therefore, the product of these two factors is an invariant for a given input size. Reduction in E_p with increase in system size can be explained as follows: At any given time step, all the PEs are working on one anti-diagonal Consecutive parallel steps involve a certain number of computation stages performed by each PE as given in (6), which is inversely proportional to the system size. As a result if the number of PEs increases, the number of stages per anti-diagonal computation reduces. Consequently the number of times each PE is activated reduces, contributing to lower overall computation energy. This is confirmed in the results shown in Figure 3.12(a). Consequently, the total energy first reduces (following the decrease in computation energy) and then increases as communication energy starts to dominate, as shown in Figure 3.12(a).

Both communication and computation time complexities of the AD approach are $O\left(\frac{(m \times n)^2}{p}\right)$. Since in our experiments, m=n, this is equivalent of $O\left(\frac{(m \times n)}{p}\right)$. Therefore, as the system size doubles, the overall time halves as shown in Figure 3.12(b). Figure 3.12(c) shows the energy-time product for the AD approach, and as can be observed the optimum number of PEs is 128.



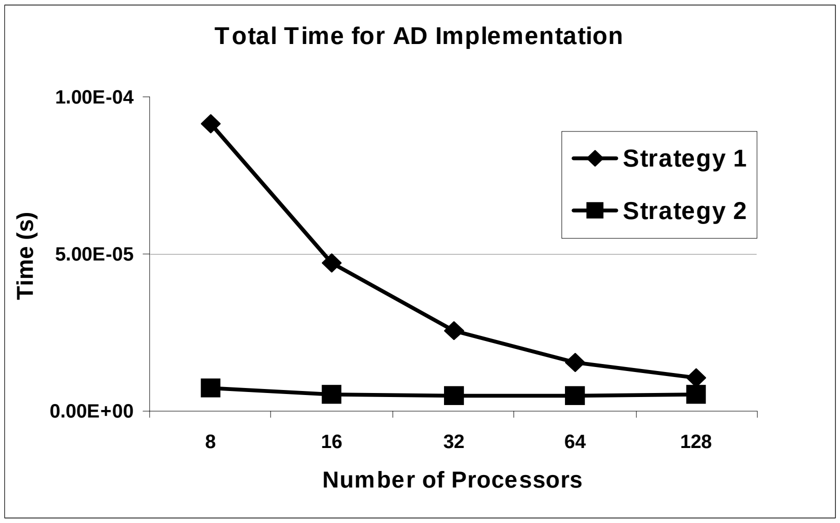


Figure. 3.13.(a) Total Time (b) Energy dissipation profiles for AD – Strategy 2.

For the second strategy, the total number of distinct boundary communications were reduced, which is given by:

$$Communications = (p-1) \times S_2 \qquad (9)$$

where S_2 is the length of one of the sequences and p is the number of processors in the network on chips. This explains the savings in the communication energy and consequently the total energy as shown in Figure 3.13 (a). The valley point in the energy curve is attained in case of the 64 PE system, and from the break-up of the total energy into the constituent components, it becomes evident that with the second way of allocating processors has computation energy as the

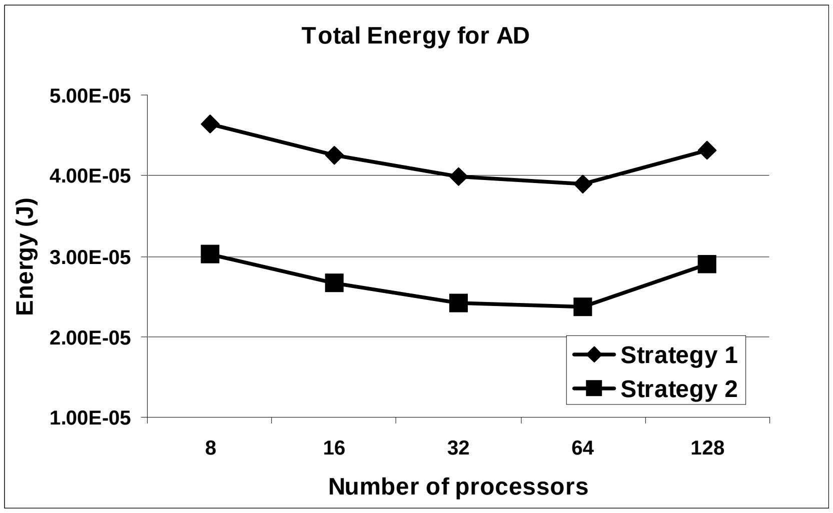


Figure. 3.14. Energy dissipation profiles for the two different AD strategies.

dominating factor. The Figure 3.14, clearly demonstrates the energy dissipation for the two different strategies, In terms of time, the total time is dominated by the share of computation, rather than communication. This explains the steady decrease in the total time on deploying larger number of processing elements for the same amount of workload. In comparison to the first strategy, the significant reduction in total time is also a result of independent computation of the cells by each processor, and communication for only the boundary cells.

Table 3.1 presents a comparative performance evaluation between the PP and AD algorithms in terms of energy dissipation and timing. It is evident that the PP approach out-performs the AD both in terms of time and energy when the system size is less than or equal to 64. But, as we increase the number of processors beyond 64, the sharp rise in the communication energy in PP attributes to rise in its total energy. Thus, for large system sizes AD approach outperforms PP in terms of energy dissipation, though it still takes more time. Among the AD strategies, the second approach is most suitable both in terms of energy and in terms of time, it is almost at par with the PP approach. For large system sizes (beyond 64), definitely this beats the PP in energy aspect as

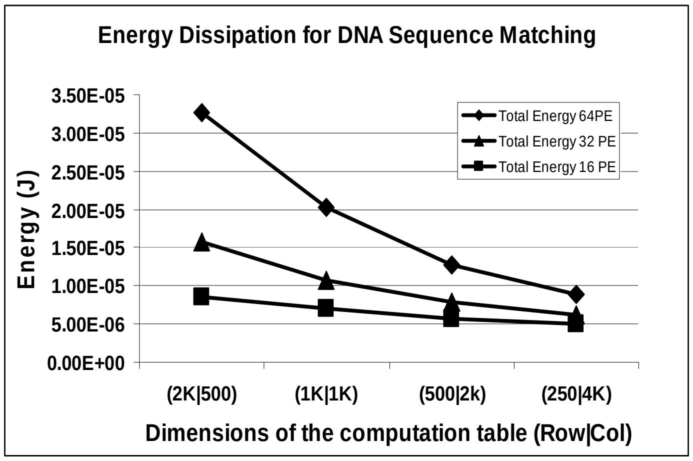
TABLE 3.1
COMPARATIVE EVALUATION OF PP AND AD SCHEMES FOR 1KX1K DATA

System Size	Energy (J)			Time (s)		
	PP	AD 1	AD 2	PP	AD 1	AD 2
8	5.57E-06	46.4E-06	30.3E-06	7.24E-06	91.3E-06	7.42E-06
16	7.01E-06	42.6E-06	26.7E-06	5.15E-06	47.4E-06	5.42E-06
32	10.7E-06	39.8E-06	24.1E-06	4.41E-06	25.7E-06	4.78E-06
64	20.3E-06	38.9E-06	23.7E-06	4.30E-06	15.3E-06	4.82E-06
128	47.2E-06	43.2E-06	29.0E-06	4.61E-06	10.4E-06	5.20E-06

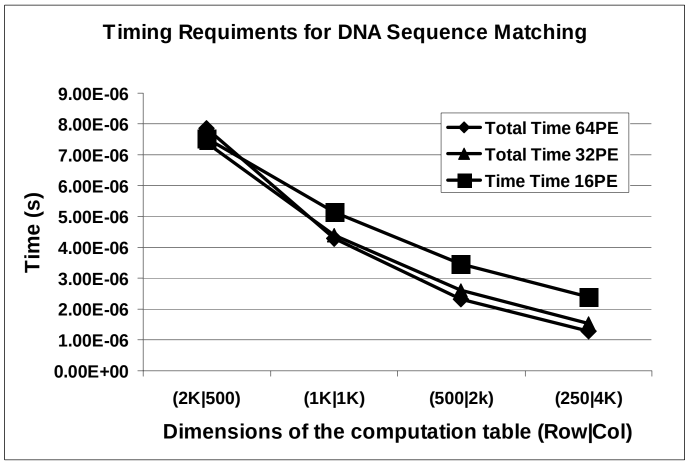
well.

3.3.6. Impact of varying string sizes

Initially, in our analysis the lengths of the two strings were maintained equal. Here, we study the effect of comparing two strings of different lengths. Figure 3.15 shows the impact of varying the string sizes on the timing and energy dissipation for the PP implementation. In order to allow a fair comparison, we varied the string sizes but keeping the total work (i.e., number of cells in the DP table) same. We considered four different combinations for s_1 and s_2. As the number of rows is decreased and/or the number of columns increased, the total time and energy both decrease.



(a)



(b)

Figure. 3.15.(a) System energy profile (b) Timing requirements plot obtained by varying the lengths of the strings using the PP implementation. The rows correspond to the characters in s_1 and the columns correspond to the characters in s_2.

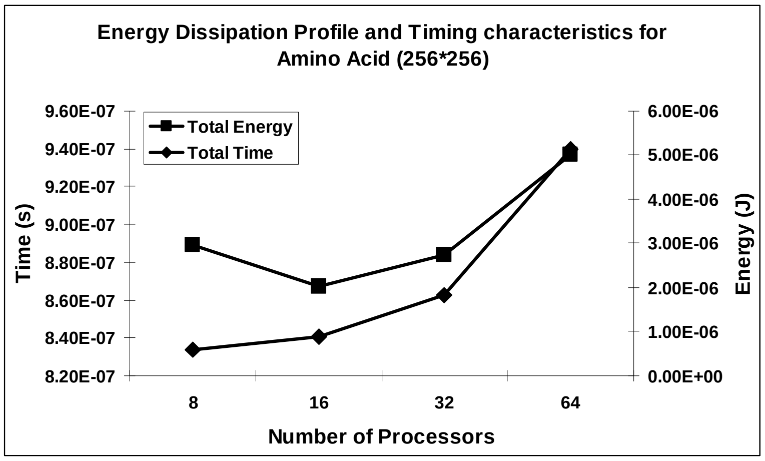


Figure. 3.16. Energy dissipation profile and Timing requirements for AA data

This can be explained by the decrease in the number of communication steps with decreasing number of rows, though the total amount of computation still remains the same. For the AD approach, the computation is always along the antidiagonal and the communication is only with the neighboring PE. Hence, there is no change in either time or energy when the string lengths were varied keeping the area of the DP table the same.

Next, we increase the number of bits from 3 to 4 for representing each of the characters of the DNA sequence to accommodate standard ambiguous character encoding. Tables 3.2 and 3.3 show the timing and energy statistics, respectively, using the PP implementation. It can be observed that the total time and the total energy increase only marginally from the 3-bit to 4-bit representation.

3.3.7. Parallel Prefix Implementation on Protein Sequence data

TABLE 3.2
TIMING COMPARISON FOR 1KX1K DATA

Number of PEs	3 bit (s)	4 bit (s)
64	4.30E-06	5.27E-06
32	4.41E-06	5.92E-06
16	5.15E-06	6.09E-06

TABLE 3.3
ENERGY COMPARISON FOR 1KX1K DATA

Number of PEs	3 bit (J)	4 bit (J)
64	20.3E-06	21.9E-06
32	10.7E-06	11.4E-06
16	7.01E-06	7.50E-06

We also undertook another case study with protein sequences, which contain one of 20 amino acid residues at each character position. In Figure 3.16, we present the energy and timing results on amino acid sequences of length 256×256 (to reflect the average length of a protein sequence). We adopted the PP algorithm (5 bit representation for each protein character), and used the PAM substitution matrix [14] for assigning the score. The trend is very similar to that for the DNA sequences of length 256×256 except for an increase in time and energy. The energy increase has been a result of the increase in computation energy due to table look-up. There is no change in the communication energy. The increase in time is also due to the increase in computation time. The same trend as DNA sequence matching is expected while implementing using AD algorithm.

TABLE 3.4
SPEEDUP OF VARIOUS ACCELERATORS OVER OUR SERIAL IMPLEMENTATION

	Intel 2.3GHz Xeon CPU	GPU	CBE	CBE	FPGA	OUR NoC IMPLEMENTATION	
						PP	AD
Time (ms)	100	1.43	0.65	17.5	1	0.00439	0.01054
Speedup over serial implementation	1	69.93	153.85	5.7	100	22779.04	9487.667

TABLE 3.5
SPEEDUP OVER EXISTING HARDWARE ACCELERATORS

Other Accelerators		FPGA [10]	CBE [12]	CBE [13]	GPU [11]
Our Implementation	PP	227.79	148	3986.3	325.74
	AD	94.87	61.67	1660.3	135.67

To assess the real benefit that can be realized using our NoC implementation we compared our run-times against other hardware accelerator implementations and our own serial implementation on a 2.3GHz Xeon CPU. The results are tabulated in Tables 3.4 and 3.5. The time needed to transfer the sequences from main memory and writing back the resultant path back to the main memory depends on the adopted interface mechanism. If the NoC-based chip is used as a co-processor on the same mother board as the main processor then the total time was about 0.096 us. This number is derived using a bus-width of 128 and a bus speed of 1333MHz. On the other hand if the PCI express 3.0 is chosen as the interfacing standard then the timing requirement is higher, which is around 2 us. The co-processor based implementation overhead is included in the timing data for our NoC implementation presented in Tables 3.4 and 3.5. As can be observed, our PP-based NoC implementation provides between 150 to 4000 fold speedup over other existing accelerators and more than well over 10^4-speedup over the serial implementation. If the PCI-based interface is used then also our implementation achieves between 100 to 2700 fold speed up over the existing accelerators.

3.4. Long Range link insertion

In PP scheme, the number of multi-stage communications is directly proportional to the number of processing elements and also to the length of one the sequences being compared. Even applying the bypass strategy, use of traditional metal wires gives rise to significant timing and power penalties. Consequently, the need for efficient long range links which can achieve the communication in a single cycle arises. This will help to reduce the power and improve the latency of communication. In this section, we evaluate performance of on-chip wireless links used as the long-range communication medium to improve the performance of the proposed NoC architecture. This exercise was undertaken specifically for solving the PP-based approach as the AD method involves neighborhood communication. Using carbon nanotube (CNT) antennas, it is possible to create on-chip wireless communication links [15, 16].

The design of the network on chip has been done in a way that we consider the neighboring communication (one-hop) to be performed by wired links, as they perform best for short range scenarios. For the longer communication (2 or more hops), the wireless links have been considered. So, every node in the network has a wireless transceiver along with the existing wired platform. In [17], it has been shown using CNT antennas 24 different frequency channels can be created. Thus, 24 frequencies can simultaneously support different wireless links in a way that a single frequency channel would be used only once per time step to avoid interference. So for system sizes greater than 24, a ‘long range wireless communication step’ has been sub-divided into multiple steps using time division multiple access for the channels. We had considered only 16 different channels for designing our point-to-point communication scheme. This has been done in accordance with the power of two scaling. In Figure 3.17, it has been shown why we require four different time sub-steps for accomplishing each of the final four time steps, of the total of six step (refer Figure 3.9) all-to-all communication for a 64 PE system. By

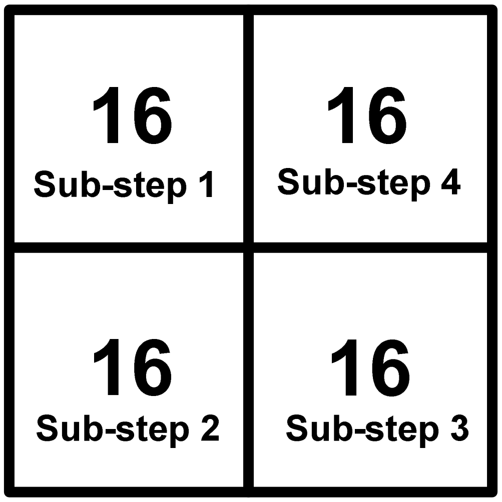
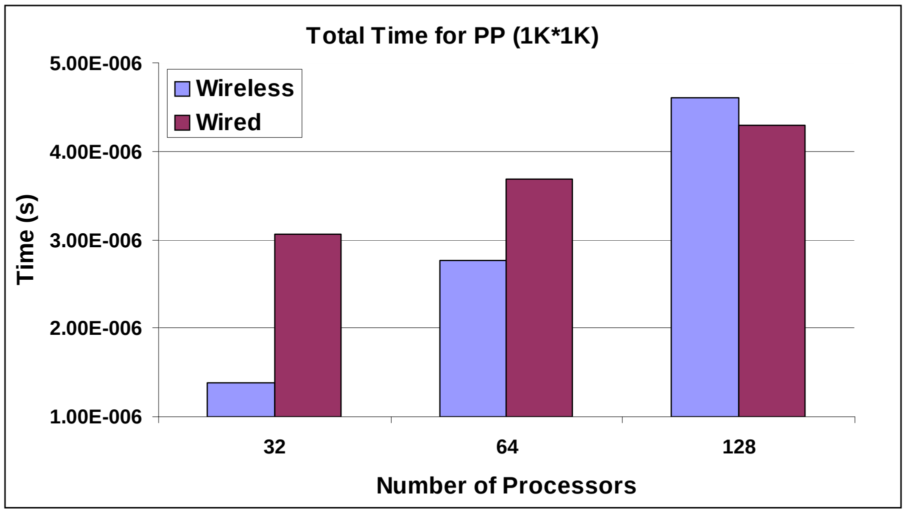


Figure 3.17. Communication sub-step allocation for the 64 PE system.

similar reasoning, for a 128 PE system, we require eight sub-steps. This multiple communication sub-steps can be attributed to the rapid increase in the total time (Figure 3.16 (a)) for large system sizes.

The Fig 3.16 represents the total energy and timing requirements for PP approach using the wireline and the wireless schemes. For larger system sizes wired network infrastructure performs better, but there is significant energy savings upon using the wireless communication fabric (Figure 3.16(b)). When considering the Energy-Delay metric, we notice that the hybrid wireless network on chip clearly emerged as the winner (as shown in Fig 3.17).



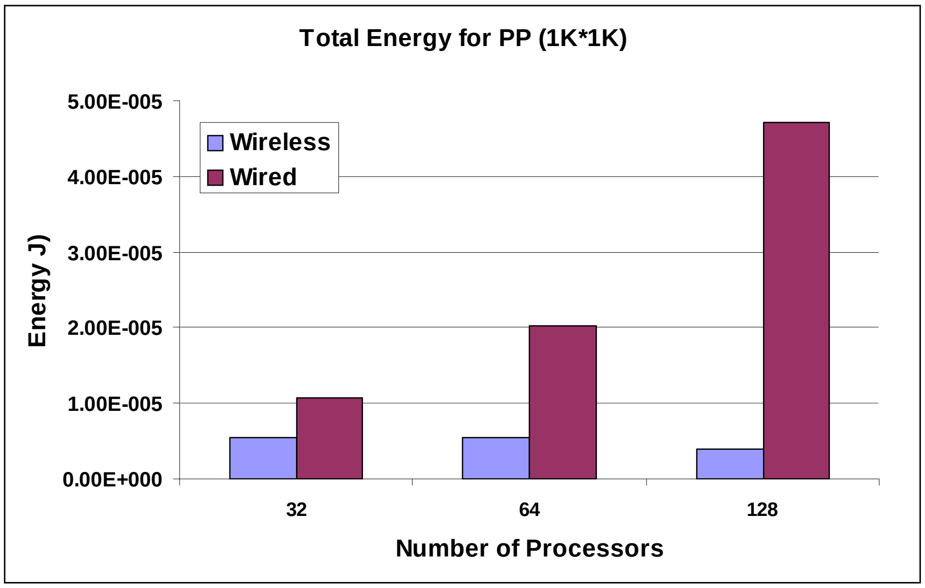


Figure 3.18.(a) Total Time (b) Energy dissipation profiles for PP upon insertion of wireless links.

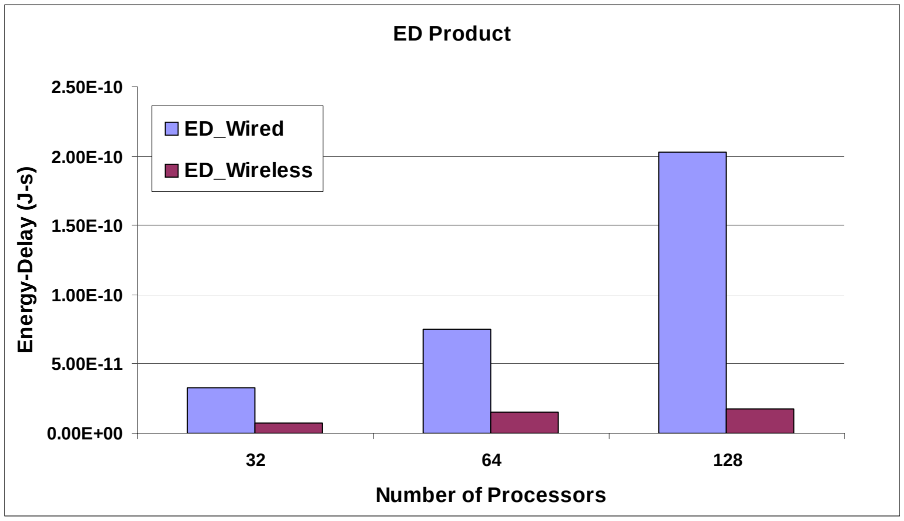


Figure 3.19. Energy Delay Product on using wireless network infrastructure

3.5. Summary

The above results go well beyond demonstrating the paradigm-shifting potential of NoC architectures over bioinformatics applications. For example, the analysis of over 28 million metagenomic sequences that took months to complete after parallelization at the coarse level [18], can be completed in a matter of days using our NoC based hardware accelerator. The NoC architecture can not only provide such high performance improvements but also, more importantly, enable solving much larger problems than was ever possible before under practical experimental settings. The widespread adoption of this design approach depends on both the design time and cost, and also the amortized cost of the total number of such systems required.

3.4 Reference

[1] T.F. Smith and M.S. Waterman, "Identification of common molecular subsequences", Journal of Molecular Biology, 1981, 147: pp. 195-197.

[2] S.B. Needleman and C.D. Wunsch, “A general method applicable to the search for similarities in the amino acid sequence of two proteins”, Journal of Molecular Biology, 1970, 48: pp 443-453.

[3] O. Gotoh. “An improved algorithm for matching biological sequences”, Journal of Molecular Biology, 1982, 162, pp. 705-708.

[4] S. Aluru et al., “Parallel biological sequence comparison using prefix computations”, Journal of Parallel and Distributed Computing, 2003, 63: pp. 264–272.

[5] E.W. Edmiston and R.A. Wagner. “Parallelization of the dynamic programming algorithm for comparison of sequences”, Proc. International Conference on Parallel Processing, 1987, pp. 78-80.

[6] X. Huang, “A space-efficient parallel sequence comparison algorithm for a message-passing multiprocessor”, International Journal of Parallel Programming, 1989, 18(3): pp.223–239.

[7] S. Rajko and S. Aluru, “Space and Time Optimal Parallel Sequence Alignments”, IEEE Transactions on Parallel and Distributed Systems, 2004, 15(12): pp. 1070-1081.

[8] A. Apostolico et al., “Efficient Parallel Algorithms for String Editing and Related Problems”, SIAM Journal on Computing, 1990, 19(5), pp. 968-988.

[9] D. Gusfield. “Algorithms on strings, trees and sequences: Computer science and computational biology”, 1997, Cambridge University Press.

[10] D. S. Hirschberg, “A linear space algorithm for computing maximal common subsequences”, Communications of the ACM, 1975, 18(6): pp.341-343.

[11] P. P. Pande et al., “Performance Evaluation and Design Trade-offs for Network on Chip Interconnect Architectures”, IEEE Transactions on Computers, 2005, 54(8): pp. 1025-1040.

[12] A.R. Butz: Alternative algorithm for Hilbert’s space filling curve. IEEE Trans. On Computers, 20: pp. 424-42, April 1971.

[13] J. M. Rabaey et al., “Digital Integrated Circuits-A Design Perspective”, Prentice Hall, 2003.

[14] Dayhoff, M.O., Schwartz, R.M., Orcutt, B.C. A model of evolutionary change in proteins. In "Atlas of Protein Sequence and Structure" 5(3) M.O. Dayhoff (ed.), 345 - 352 (1978).

[15] Amlan Ganguly, Kevin Chang, Sujay Deb, Partha Pande, Benjamin Belzer, Christof Teuscher, “Scalable Hybrid Wireless Network-on-Chip Architectures for Multi-Core Systems”, IEEE Transactions on Computers (TC), August, 2010, URL: http://www.computer.org/portal/web/csdl/doi/10.1109/TC.2010.176

[16] K. Kempa, et. al., “Carbon Nanotubes as Optical Antennae”, Advanced Materials, 2007 vol 19, pp. 421-426.

[17] B.G. Lee et. al., “Ultrahigh-Bandwidth Silicon Photonic Nanowire Waveguides for On-Chip Networks”, IEEE Photonics Technology Letters, March 2008, vol. 20, no. 6, pp. 398-400.

[18] S. Yooseph et al., “The Sorcerer II Global Ocean Sampling Expedition: Expanding the Universe of Protein Families”, Public Library of Science Biology, 2007, 5(3):e16doi:10.1371/ journal.pbio. 0050016.

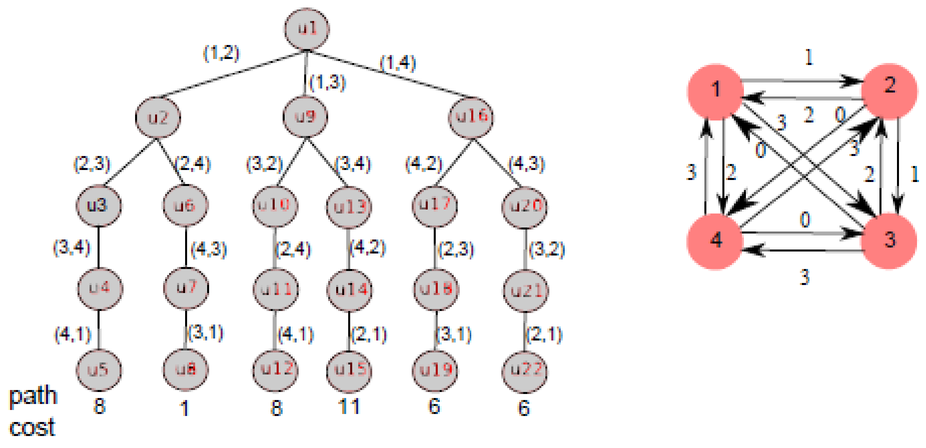
Chapter 4

NoC Architecture for Phylogenetic Reconstruction

In this chapter, we undertake design and performance evaluation of NoC architectures for phylogenetic reconstruction. We consider the Maximum Parsimony (MP)-based phylogeny, which depends on finding the breakpoint median, when given a set of species. The breakpoint median reduces to one of solving multiple instances of the Traveling Salesman Problem (TSP), which is a classical NP-complete problem in graph theory. In the following subsections, we first explain the problem, the algorithmic details, followed by the design and implementation of a multi-threaded software program for modeling the communication events, and then the network on chip based platform for solving TSP.

4.1 Algorithm for Traveling Salesman Problem

In this section, we present the core computation steps of the branch-and-bound heuristic to solve TSP [1] 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) [2]. The output is a least cost cyclic tour that traverses all vertices exactly once.



(b) (a)

Figure 4.1. An example showing (a) an input graph and (b) the exhaustive search tree corresponding to the input graph. 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.

Given this input graph *G*, the solution space can be represented by a conceptual computation tree. An example is shown in Figure 4.1. 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 $(i,j) \in E$, 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 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 Figure 4.2.

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 lower bound computed (*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.

Lower bound calculation. We use the method shown in [1] for lower bound computation at each tree-edge. An *m x 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 ∞;

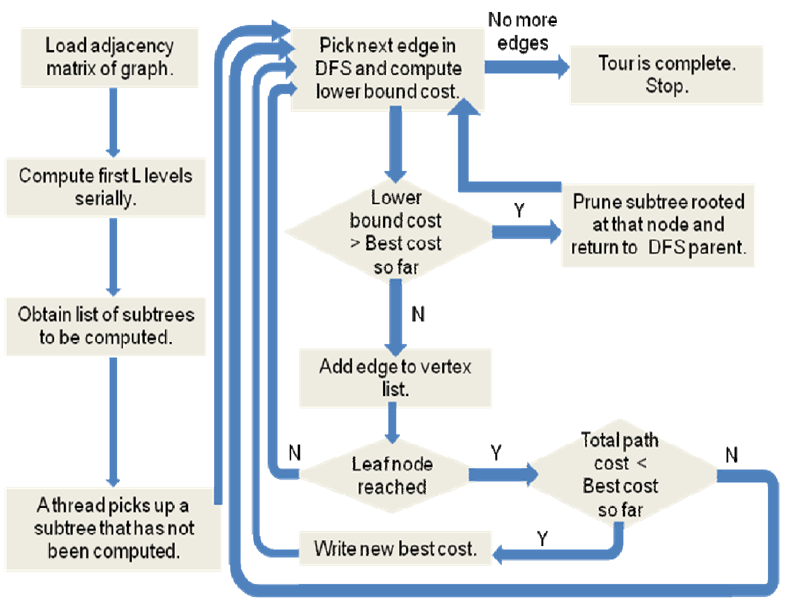

Load adjacency matrix of graph.
Compute first L levels serially.
Obtain list of subtrees to be computed.
A thread picks up a subtree that has not been computed.
Pick next edge in DFS and compute lower bound cost.
No more edges
Tour is complete. Stop.
Lower bound cost > Best cost so far
Y
Prune subtree rooted at that node and return to DFS parent.
N
Add edge to vertex list.
N
Leaf node reached
Y
Total path cost < Best cost so far
N
Y
Write new best cost.

Figure 4.2. Flow diagram explaining branch-and-bound algorithm for solving breakpoint median problem

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: $lbc(v) = lbc(u)+R[i,j]+adjCost$.

4.2. Network on Chip Design

The problem of MP phylogenetic reconstruction using branch-and-bound heuristics 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 remainder of this section details the design of the PE, switches and the communication fabric.

4.1 PE Design

The PE has a pipelined architecture optimized to handle the computation along an edge as per the algorithm described in Sec. 3. 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 PE has an integer datapath because breakpoint median computation for MP phylogenetic 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.

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 Figure 4.3. Subsequent references to the sub-blocks in parentheses (e.g. ρ, φ, etc.) in this sub-section refer to this figure.

4.1.1. Reduction block. This block (ρ) carries out the matrix reduction operation described in Sec. 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*

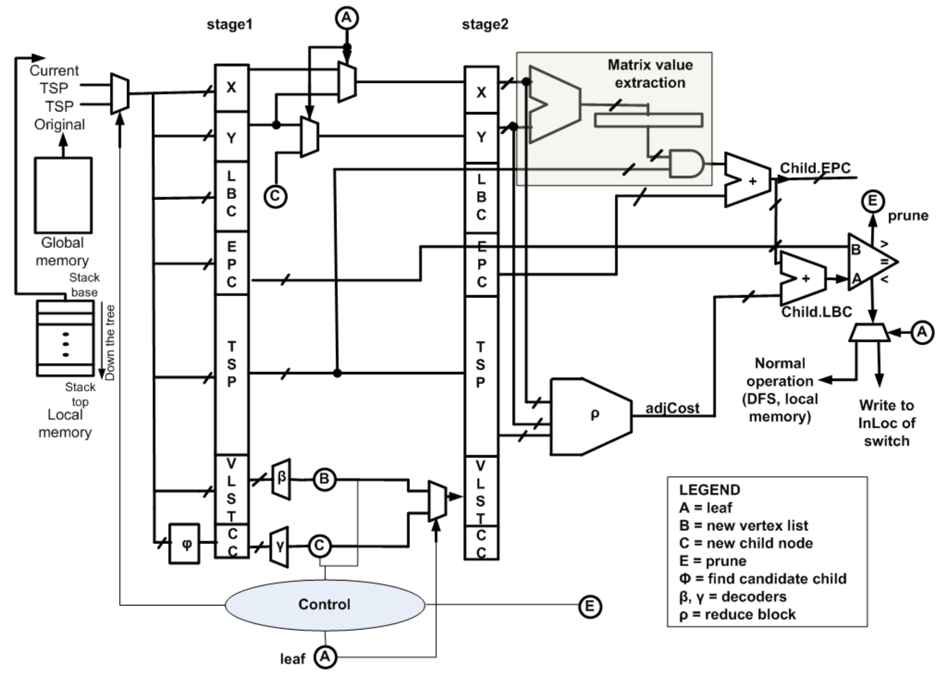


Figure 4.3. Internal architecture of processing element (PE) for edge reduction.

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 *Peripheral control logic* below for details on the operations upto this stage) and the adjacency cost *adjCost* is obtained. Figure 4.4 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 under the current implementation.

4.1.2. 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

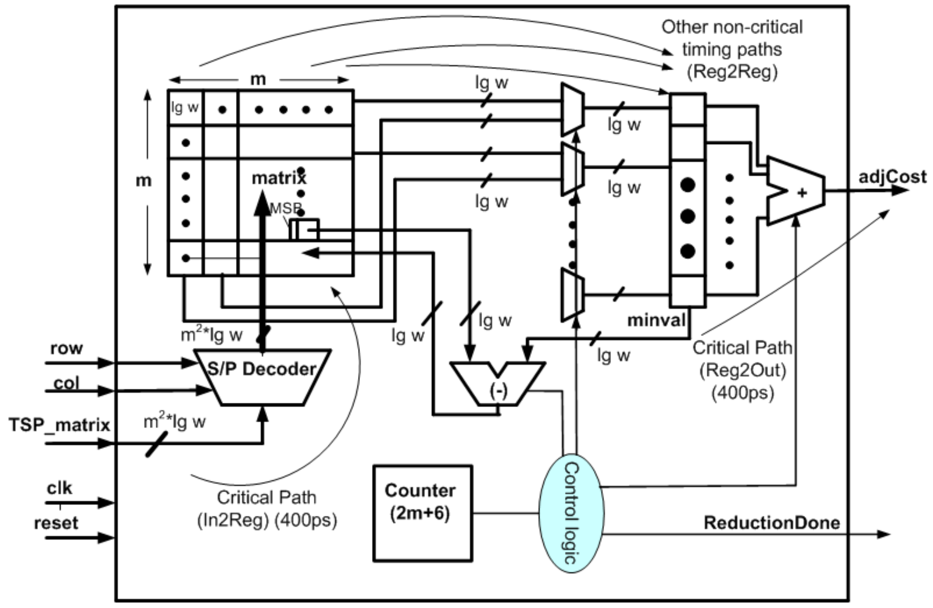


Figure 4.4. Internal architecture of reduction block (ρ) for linear-time matrix reduction.

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.

4.1.3 Memory. There are two logical divisions in on-chip memory – *global* and *local*. However, all memory is physically distributed across all PEs. The *global* memory in a PE stores the TSP matrix that represents the subtree assigned to that PE. The *local* memory is implemented as a stack. During DFS, the new vertex data (path cost, vertex list) is pushed into the stack (Figure 4.3). 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. Every PE has a *m*-sized *local* memory stack.

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.

4.2 Network Design

We explored two different kinds of network architecture – a mesh, shown in Figure 4.5 (a) and a quad-tree, shown in Figure 4.5 (b). 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 sytem size or the number of nodes (PEs). The same for a quad-tree increases as $O(\log_4 N)$. As the *best cost* is written to all PEs except for the originating PE, the mode of communication for our application involves some form of broadcast. 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 1 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. A quad-tree has an advantage over a 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 comparisons are provided in Sec. 5.

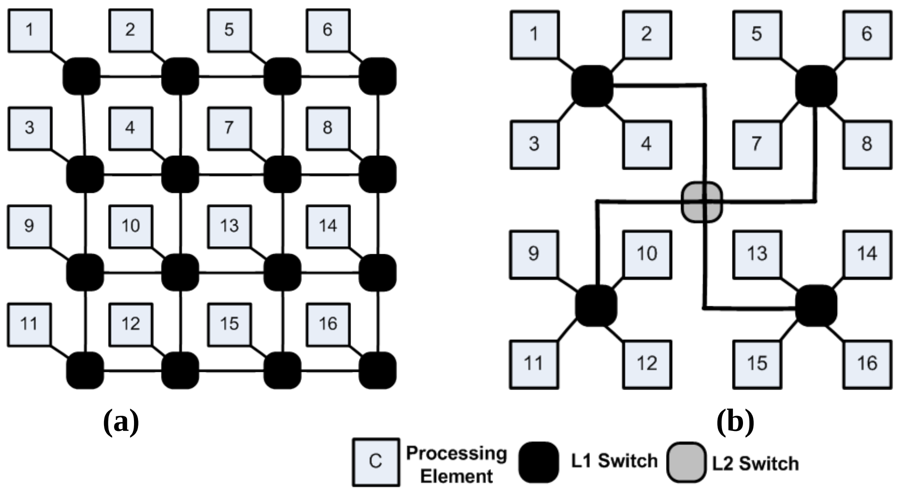
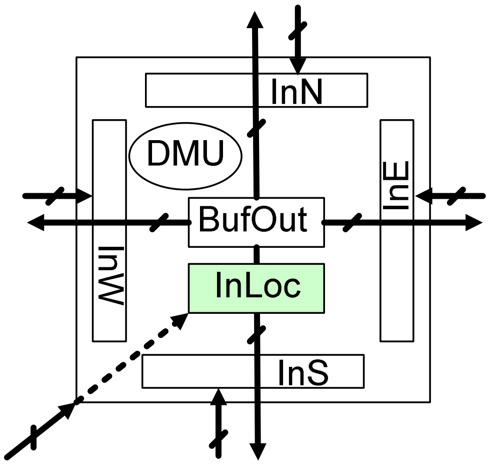
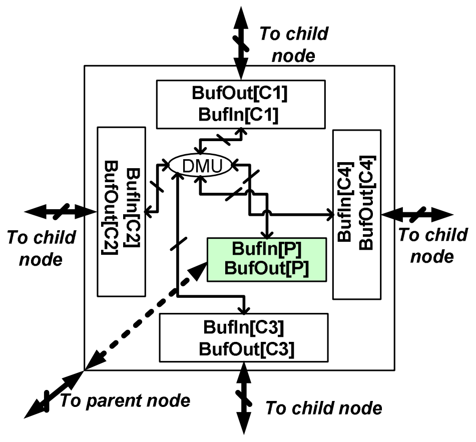


Figure 4.5. (a) Mesh NoC architecture (b) Quad-tree NoC architecture.



(a)



(b)

Figure 4.6. Internal architecture of switch for (a) mesh and (b) quad-tree.

4.3 Switch Design

Different switches are designed for each of the two network architectures explored. The switch and the PEs run on the same system clock. Since we have a pipelined (switch-to-switch) communication technique, a globally synchronous NoC does not pose a problem with scalability.

4.3.1. Mesh. A typical switch that is used on a mesh is shown in Figure 4.6 (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 4.7 shows a 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 consideration or not. The data is considered if the control bits are *DOTX* or *TRWL* but not if they are *NOTX* or *NORETX*.

4.3.2 Quad-tree. There are different levels of switches for this network architecture. The leaf level switches (refer to Figure 4.5(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 Figure 4.6(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.

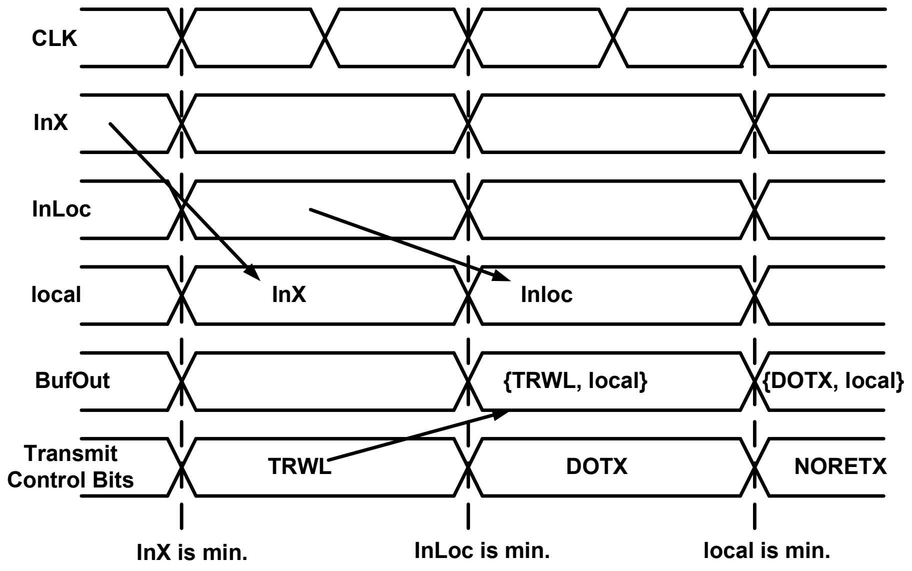


Figure 4.7. Timing diagram showing typical scenarios encountered in a mesh switch.

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) cost, PC_i for a decision cycle comes from $i \in L$ i.e., $PC_i < PC_j \forall j \neq i, j \in L$. Then, we have

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

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

4.4 Communication Protocol

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 the previous sub-section, 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*. 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.

4.5. Experimental Results

4.5.1 Experimental Setup

In order to completely model the performance of the network on chip based platform, first a multithreaded software suite was developed using pthread library in C. The prime objective was to model the communication events occurring during the entire solution for the TSP. Also, the number of reductions performed by each individual thread was noted, along with the successful write updates to the global best score. The power dissipation by a single processing element is directly proportional to the number of reductions performed by it, and the number of reads and writes of the global best cost location gives an estimate of the network traffic event and the interconnect power. The load balancing issue was addressed by first reducing the top few levels serially, and then from the list of unsolved subtrees, the individual subtrees were picked up by the available threads. The number of levels reduced serially was set such that the list of available subtrees (in that level) is much larger than the total number of threads. This is clearly shown in Figure 4.8.

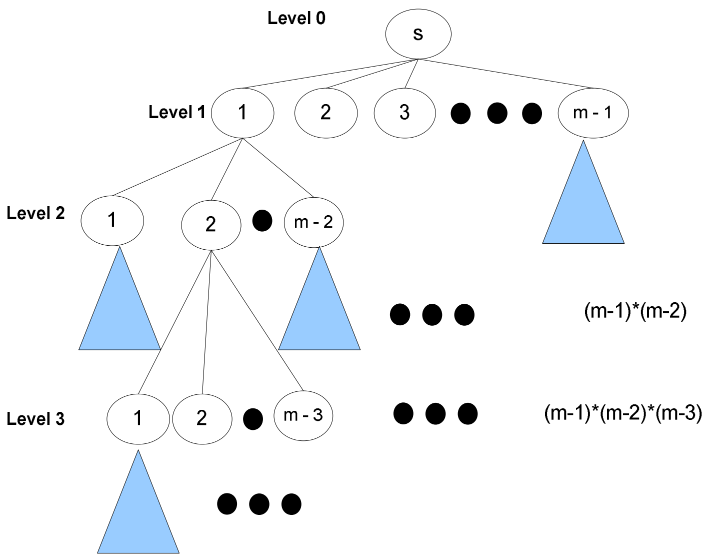


Figure 4.8. Diagram showing the number of subtrees generated by "cutting" the solution-space tree at different levels.

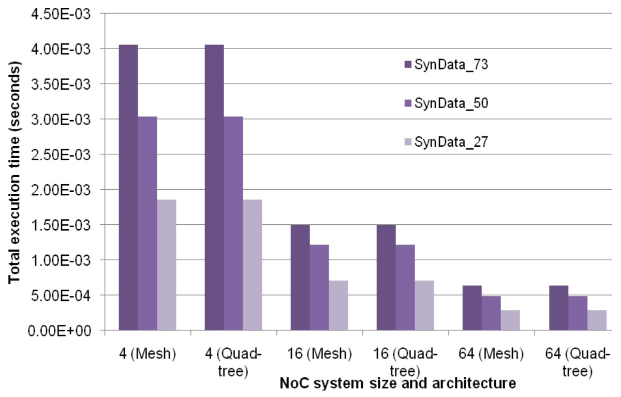
The performance evaluation of the NoC was carried out from the timing and power perspectives during phylogenetic reconstruction with varying data sets. Different parameters associated with the NoC are as follows. The system size, *N*, is the number of PEs in the NoC. *N* was set to 4, 16 and 64 for evaluating 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 Sec. 5.3). Note that the value of *m* affects the size of the datapath and the memory requirements in the PE as per the discussion in Sec. 4. 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.

The PEs and the switches in the NoC were implemented by synthesizing Verilog RTL using Synopsys Design Compiler and 65 nm library [3]. The pipelined design could sustain a clock frequency of 2.5 GHz in the PEs and switches. This was verified with *m*=110 and higher. Power numbers for PEs and switches have been reported from Synopsys Power Compiler using the same library [3]. Interconnect characteristics were determined using Cadence Spectre. Wire capacitance information extracted from layout was used to determine delay and energy dissipation of interconnects. Both mesh and quad-tree architectures were considered for performance evaluation.

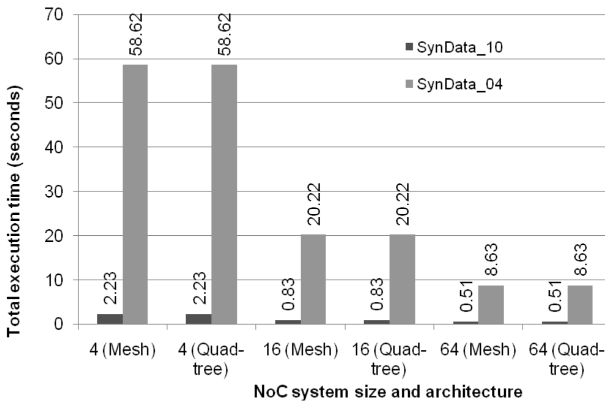
GRAPPA [4] was used as the software benchmark. It is a standard and widely used serial program for MP phylogenetic analysis. GRAPPA was run on a quad-core 2.40 GHz Intel Xeon E5530 processor with 16 GB of RAM. The run-time measured through GRAPPA served as the basis in speedup calculations. Specifically, speedups reported are calculated as the ratio of GRAPPA run-time over the total execution time on an *N*-PE NoC.

5.2 Performance on synthetic data

Five synthetic data sets were generated and used as input. Each input consisted of three genomes



(a)



(b)

Figure 4.9. Total execution time in hardware for (a) *SynData_73*, *SynData_50* and *SynData_27* and (b) *SynData_10* and *SynData_04*

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*.

5.2.1 Timing Performance. Figures. 4.9(a) and 4.9(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

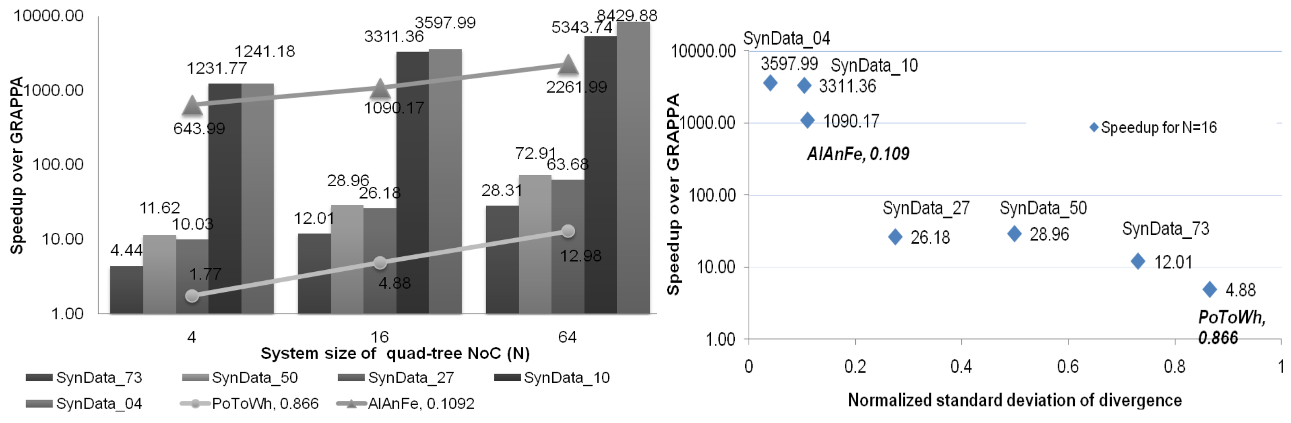


Figure 4.10. (a) Absolute speedup over serial GRAPPA (b) Variation of speedup with skew of input data on quad-tree NoC with N=16.

the total computation and communication cycles spent in the NoC and the time required to load the data on the NoC using PCI-X. 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 Sec. 5.4), 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 1), which contributes a small fraction to the total execution time in this case.

Figure 4.10 (a) shows the speedup over GRAPPA using a quad-tree for these inputs. Since speedup is the ratio of GRAPPA's serial 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 Figure 4.10 (b), 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 [5].

Note that the synthetic data encompass almost the full range of possible inputs, with Δ

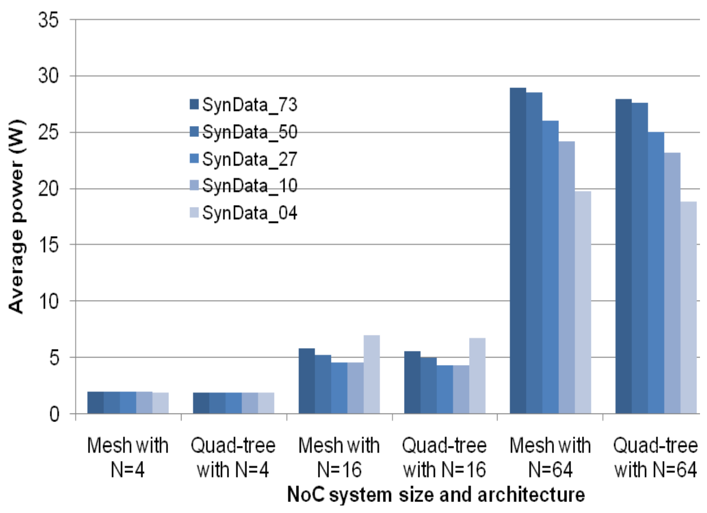
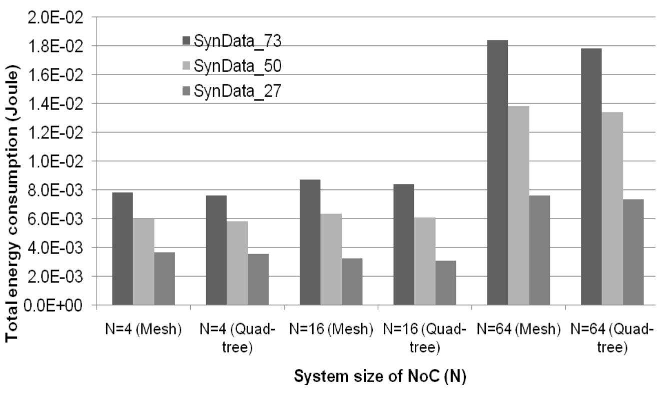
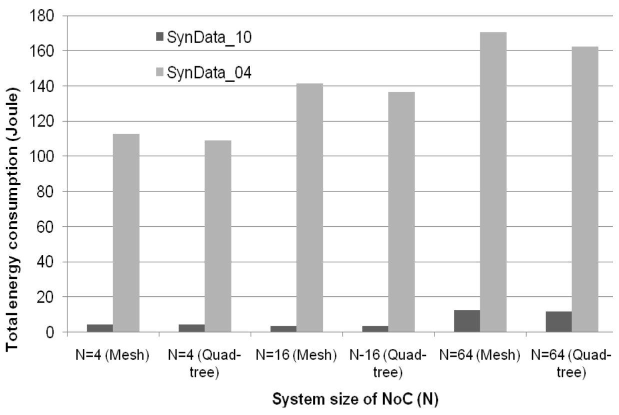


Figure 4.11. Full chip power consumption across various inputs, network architectures and system sizes

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 Sec. 5.3, 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.

5.2.2 Energy Performance. Several measures were used to evaluate the energy performance of the NoC. The average full chip power consumption for mesh and quad-tree NoCs for *N*=4, 16 and 64 is shown in Figure 4.11. 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-based chip 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





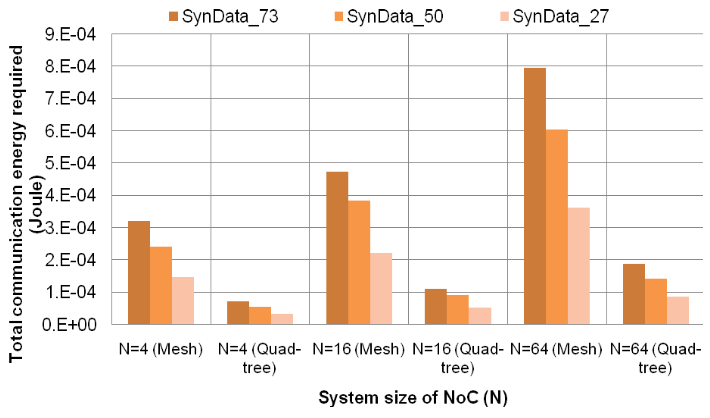
(a) (b)

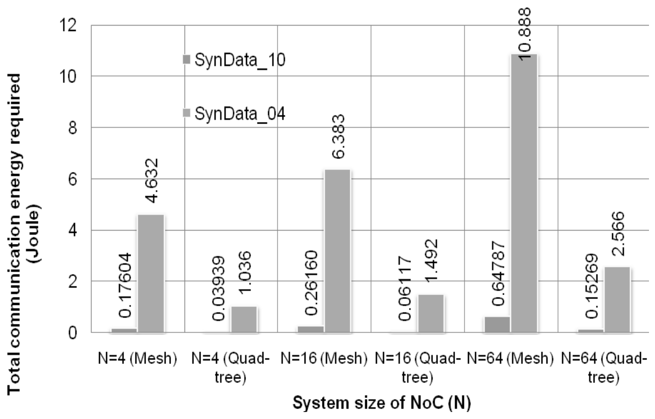
Figure 4.12. Full chip energy consumption across different inputs

more accurate rubric is the total energy consumption, shown in Figure. 4.12(a) and 4.12(b). Although these figures show the advantage of quad-tree over mesh in terms of energy performance, comparing only the communication energy consumptions in Figure. 4.13(a) and 4.13(b) further highlights this. Quad-tree consistently outperforms mesh by consuming around 75% less communication energy. Both average full chip 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 seen from Figure. 4.14 (a) and 4.14 (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.

5.3 Performance on 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 [6]. One input (*PoToWh*) consisted of the chloroplast genomes of *Solanum tuberosum* (potato, 141 genes), *Solanum lycopersicum* (tomato, 130 genes) and *Triticum*





(a) (b)

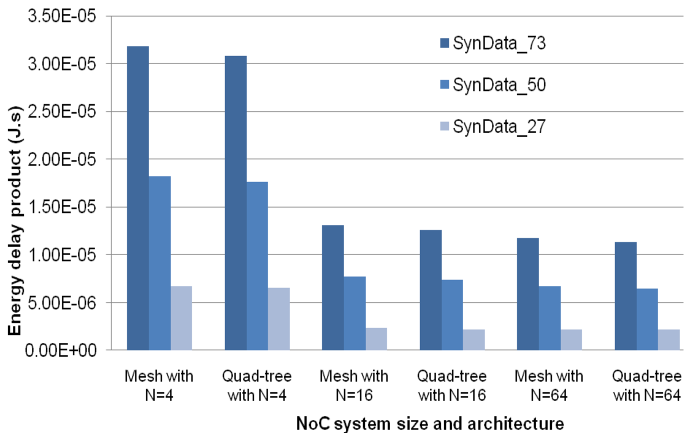
Figure 4.13. Communication energy expended across different inputs

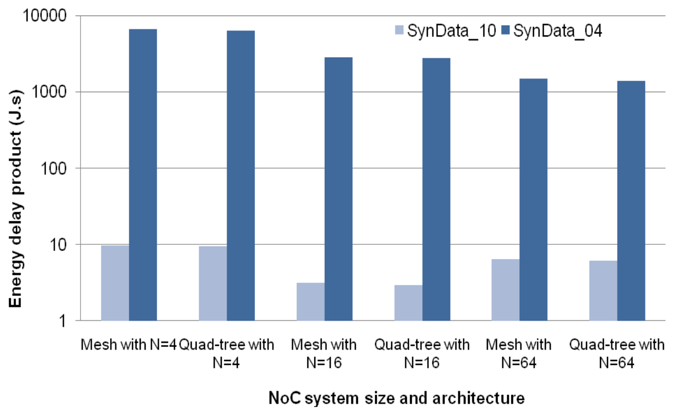
aestivum (bread wheat, 137 genes). The other input (*AlAnFe*) consisted of chloroplast genomes of *Chlamydomonas reinhardtii* (a unicellular green alga, 109 genes), *Brachypodium distachyon* (purple false brome grass, an angiosperm, 133 genes) and *Adiantum capillus-veneris* (black maidenhair fern, 130 genes). These genomes were preprocessed with Mauve [7] 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 Figure 4.9.(a) and (b) shows the speedup correlation with synthetic data having similar values of Δ.

As mentioned in Sec. 5.1, speedup is calculated as serial GRAPPA run-time divided by the total execution time on the NoC. As explained later in Sec. 5.4, 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 and have the same ratio. In comparison, the GRAPPA run-times are 5.55 ms and 9.22 s respectively. Next, we turn our attention to the variation of speedup with increasing *N*. It can be seen from Figure 4.9 (a) 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 2 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 (Figure. 4.14(a) and 4.14(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.

TABLE 4.1. 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



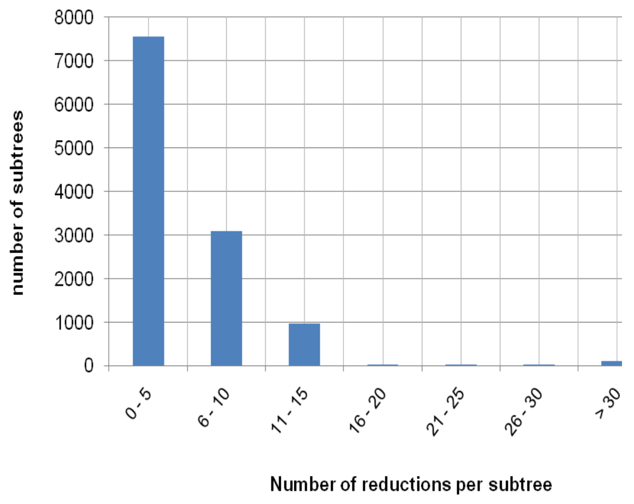


(a) (b)

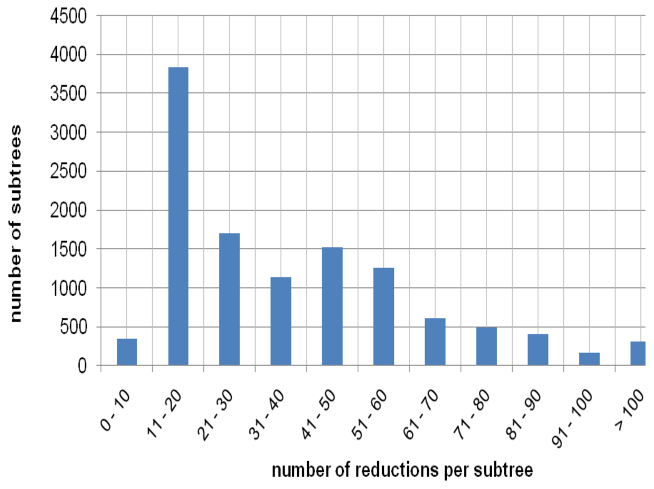
Figure 4.14. Variation of energy-delay product across inputs

5.4 Performance tradeoffs

The PE that finishes its share of reduction computations last limits the performance of the entire chip. 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 finshed computing the subtree exhaustively or when it has pruned it. This results in each subtree contributing to a different number of reductions and each PE computing a different number of subtrees.



(a)



(b)

Figure 4.15. Histogram of number of number of reductions per subtree for (a) *PoToWh* and (b) *AlAnFe*

Experiments showed that the load distribution among PEs was more even when the number of subtrees in the common pool was 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 (Figure 4.7). So it appears that “cutting” the tree at a lower level generates more subtrees, helping to balance load and thereby ensure maximum achievable parallel speedup. Now, there is an overhead involved in loading the entire set of subtrees to the chip using an interface like PCI-X. This increases with the amount of data that needs to be transferred, which increases with the number of subtrees. There is also a constant time overhead incurred when each PE picks a subtree at run-time. These overheads become prohibitively high when the number of subtrees is large and they mask the gains achieved in computation speedup. We resolved this tradeoff by chossing to “cut” the tree at level 2, which generated 109*108 subtrees and yet kept the overhead to a manageable amount. Note that 109*108 is much larger than the largest system size (number of PEs, *N*=64) we experimented with, which led to a balanced load distribution.

3.5. Summary

In this chapter we have proposed and implemented a network on chip based hardware accelerator for solving the TSP, which is the main computation kernel for MP-phylogenetic reconstruction. We have demonstrated significant improvements in the performance as compared to the currently existing solutions In terms of the communication network infrastructure, we had considered mesh and quad-tree topologies, and showed how quad-tree performs better in terms of energy dissipation.

As part of this work, we had designed an architecture of the reduce block, which solves $O(N^2)$ computation in $O(N)$ time, using micro-level parallelism. It can be concluded that in future, such a NoC based platform can be adopted for energy and time efficient computation of scientific problems, which are NP complete.

4.5 Reference

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

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

[13] Circuits Multi-Projects (http://cmp.imag.fr/). Last date accessed: 20 Feb 2010.

[14] GRAPPA home page (http://www.cs.unm.edu/~moret/ GRAPPA/). Last date accessed: 16 June 2010.

[15] J. Bakos and P. Elenis, “A Special-Purpose Architecture for Solving the Breakpoint Median Problem,” IEEE Transactions on Very Large Scale Integration Systems, vol. 16, 2008, pp. 1666-1676.

[16] NCBI database for eukaryotic organelles (http: //www.ncbi.nlm.nih.gov/ genomes/genlist.cgi?taxid=2759&type=4&name=Eukaryotae%20Organelles). Last date accessed: 19 May 2010.

[17] Genome Evolution Laboratory – Mauve Genome Alignment Software (http://asap.ahabs.wisc.edu/mauve/). Last date accessed: 19 May 2010.

Chapter 5

Conclusions and Future Work

In this chapter, we summarize the contributions made in this dissertation and point to the possible future investigations emanating from this research endeavor.

5.1 Conclusions

The advent of modern multi-core chips has opened up possibilities of designing very efficient hardware accelerators for various bio and scientific computing problems. The primary challenges in this direction being efficient partitioning of the computational workload, scalability of the solution, power dissipation and performance. The major contribution of this thesis is the design and performance evaluation of multi-core-based hardware accelerators for a suite of biocomputing applications. Both the problems addressed in this dissertation belong to the class of combinatorial optimization, but sequence alignment is intrinsically data intensive in nature, whereas the breakpoint median in phylogenetic reconstruction is compute intensive. For sequence alignment work, we implemented a novel network on chip based solution, where both the architecture of the processing elements and interconnect infrastructure has been tailored pertaining to the target application. Two different space and time optimal algorithms have been considered in this respect, both inherently being fine grain parallel. Due to the fine grain nature of the computation, only a single integer is communicated at every communication stage. This necessitates intelligent redesign of the traditional NoC switch. The proposed switch architecture is simple yet effective, reducing the energy dissipation significantly. The entire communication event, which is an all-to-all event, was divided into multiple steps, some of which involved multi-hop communication. Unless long range links are introduced, the multi-hop communication consumes multiple clock cycles. It becomes prohibitive in terms of performance as communication events become the bottleneck. They occur very frequently and are directly proportional to the length of one of the strings. To achieve performance improvement, first long range links were introduced, which reduced the time considerably but the power dissipation increased a little.

In the phylogenetic reconstruction work, the principal bottleneck has been memory, as the recursive depth-first-search procedure requires saving a complete matrix at every stage of the computational search space. The alternate solution is re-computation of the cost metric at every stage, which would incur significant penalty in time. The currently designed architecture handles only genomes of maximum length 128, as it uses the stack implementation for storing the matrices. In its current state, the design has been done not in terms of the accommodating large genomes, but rather addressing the problem of studying larger number of smaller genomes. This is currently a more challenging problem for biologists as study of bacteria and virus (with smaller number of genes but large number of taxa) is becoming unmanageable, as these organisms perform mutation and cross-over operations very frequently.

5.2 Future Directions

This research can be extended not just in the direction of porting new scientific applications and reducing the power dissipation and offering better performance compared to the existing solutions, but in exploring efficient architectural innovations towards even higher levels of integration and superior performance. The research pertaining to this thesis can be carried forward in the following directions:

5.2.1 Sequence Analysis

Our research has also laid out a design template for the future development of new acceleration models for other related applications in bioinformatics. For instance, the BLAST algorithm [1], which is an approximation method for computing sequence alignments, is a popular tool for detecting performing sequence database searches. Owing to large sequence database sizes, accelerating BLAST search operations is performance-critical. Nevertheless, the underlying algorithm in BLAST also uses the Smith-Waterman algorithm, while also implementing a prefiltering process prior to computing alignment using a string look-up table data structure. Consequently, an off-shoot of our research could be that NoC can be explored as a viable means for acceleration for BLAST as well. Before such a project is undertaken, however, a feasibility study should be conducted to assess both the quality degradation that is possible due to approximation, along with the performance impact due to implementing additional string data structures.

5.2.2 Prototyping

The study undertaken in this thesis is principally simulation based, where a software driver has been used for providing the event statistics and hardware has been designed and synthesized using Synopsys Design Vision. To reinforce the findings of this work, real prototyping is an inevitable direction. The recent multi-core platforms such as Intel single chip cloud computer comprising of 48 cores [2], and Tilera NoC are two such network on chip based real platforms where the applications can be prototyped. The single ship cloud incorporates technologies intended to scale multi-core processors to 100 cores and beyond, such as an on-chip network, advanced power management technologies and support for “message-passing.” Architecturally, the chip resembles a cloud of computers integrated into silicon. The novel many-core architecture includes innovations for scalability in terms of energy-efficiency including improved core-core communication and techniques that enable software to dynamically configure voltage and frequency to attain power consumptions from 125 W to as low as 25 W.

The Tilera platform consists of an array of 16 to 100 general-purpose processor cores, where each core is 64-bit VLIW processor tiles. There is a three-way pipeline with up to three instructions per cycle. Amongst the many other salient features, it claims a power efficient inter-tile communication and also idle tiles can be put to low power sleep mode.

Once our applications are efficiently ported to these platforms, it can truly unveil the potential of NoC based solutions compared to the other existing hardware platforms. For complete study, exhaustive experimentation in this direction is need.

5.2.3 Maximum likelihood & Bayesian Inference

The other probabilistic strategies used by biologists for phylogenetic reconstruction are Maximum Likelihood (ML) are Bayesian Inference. Research in the direction of hardware acceleration can be extended by considering these as target applications.

5.2.4 Hardware multithreading

For the sequence alignment case, in both the parallel prefix and anti-diagonal algorithms, there is a computation phase followed by a communication phase. The performance can be further improved if these interleaved operations are executed using multiple hardware threads. This would eliminate any idle state, where either the computation or the communication resources are waiting for the other phase to finish. This can be considered as a simple latency hiding strategy, which can significantly improve the amortized speedup. The challenge in this approach is the increase in the complexity of the processing element, which significantly increases the area overhead.

5.3 Summary

NoC has emerged as an enabling solution for integration of huge number of embedded cores on a single die. As many hardware NoC platforms are now becoming a reality; the scientific computing domain, especially computational biology can completely leverage the potential of this novel technique by efficiently partitioning the problem, and mapping it to these platforms. We have demonstrated about how custom designing a NoC can achieve several orders of performance improvement over other existing hardware acceleration schemes.

5.4 Reference

[1] S.F. Altschul et al., "Basic Local Alignment Search Tool", Journal of Molecular Biology, 1990, 215, 403-410.

[2] http://techresearch.intel.com/ProjectDetails.aspx?Id=1.

Appendix A

Publications

Following is a list of publications published in reputed journals and conferences during the course of this research.

Journals:

1. **Souradip Sarkar**, Gaurav Ramesh Kulkarni, Partha Pratim Pande**,** Ananth Kalyanaraman, "Network-on-Chip Hardware Accelerators for Biological Sequence Alignment", *IEEE Transactions on Computers***,** 2010**,** 59(1): 29-41.

2. Turbo Majumder, **Souradip Sarkar**, Partha Pratim Pande, Ananth Kalyanaraman, "NoC-Based Hardware Accelerator for Breakpoint Phylogeny", *IEEE Transactions on Computers* , 2010 (under review).

Conferences:

1. Turbo Majumder, **Souradip Sarkar**, Ananth Kalyanaraman, Partha Pratim Pande, "An Optimized NoC Architecture for Accelerating TSP Kernels in Breakpoint Median Problem" *Proceedings of 21st IEEE International Conference on Application Specific System Architectures and Processors,* ASAP, 2010: pp. 89-96.

2. **Souradip Sarkar**, Turbo Majumder, Ananth Kalyanaraman, Partha Pratim Pande, "Hardware accelerators for Biocomputing: A Survey" *Proceedings of 2010 IEEE International Symposium on Circuits and Systems,* ISCAS, 2010: pp. 3789-3792.